The Pennsylvania State University
The Graduate School

EXPERIENCE-BASED CYBER SECURITY ANALYTICS

A Dissertation in
Computer Science and Engineering
by
Po-Chun Chen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2011

The dissertation of Po-Chun Chen was reviewed and approved* by the following:

John Yen
University Professor of Information Sciences and Technology
Professor of Computer Science and Engineering
Dissertation Advisor, Chair of Committee

Peng Liu
Professor of Information Sciences and Technology
Professor of Computer Science and Engineering

C. Lee Giles
David Reese Professor of Information Sciences and Technology
Professor of Computer Science and Engineering

Prasenjit Mitra
Assistant Professor of Information Sciences and Technology
Assistant Professor of Computer Science and Engineering

Runze Li
Professor of Statistics

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

As the demand for computational resources and connectivity increases and contemporary computer network systems become more complex, the management of cyber security is progressively becoming a serious issue.

Cyber situation recognition is a challenging problem, particularly when the network size is large. The amount of data produced by existing intrusion detection tools and sensors usually significantly exceeds the cognition throughput of a human analyst. In attempting to align a huge amount of information and the limited human cognitive load, a critical disconnection between human cognition and cyber security tools has been identified. Although the problem of cyber intrusion detection has been studied from several perspectives using various approaches, the key component to bridging the gap between existing tools and human analysts' experiences is missing. A method to capture and leverage cyber security expertise for situation recognition from a high-level viewpoint on the entire network is important, but it is rarely mentioned in the literature.

The goal of this research is to address the problem of cyber intrusion recognition from the viewpoint of leveraging cyber experts' experiences and reflections. We developed a systematic approach to capture and utilize experiences and reflections of security analysts to enhance cyber situation awareness.

The contributions of the research include: 1) proposing an approach to enable systematic capture of experience and reflection of cyber security analysts; 2) enhancing the recognition of cyber situations using the captured experiences of cyber security analysts; 3) providing a knowledge-based strategy for relaxing the constraints of Horn logic-based experience patterns to enhance their utilization; and 4) demonstrating the benefit of experience-based cyber situation recognition through simulations.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my greatest appreciation to my advisor, Dr. John Yen, for his guidance and constant support throughout the years I have been at Penn State. I have learned a lot from him, and it has been my honor to work with him.

I would also like to thank the members of my doctoral committee, Dr. Peng Liu, Dr. Prasenjit Mitra, Dr. C. Lee Giles, and Dr. Runze Li, for providing valuable advice and comments on my research and dissertation. I would like to give my gratitude to Dr. Peng Liu and Dr. Tracy Mullen for sharing their knowledge and expertise while we worked together on the MURI project. I also gratefully thank Dr. Xiaocong Fan and Dr. Dinghao Wu for providing very helpful opinions on my research over the years. I am also grateful to Dr. Yanxi Liu for guiding me to explore a different research field and broaden my knowledge.

I want to give special thanks to my co-workers and colleagues at the Laboratory for Intelligent Agents, including Dr. Shuang Sun, Dr. Shizhuo Zhu, Guruprasad Airy, Sooyoung Oh, Baojun Qiu, Kang Zhao, Eun Yeong Ahn, Hyun-Woo Kim, Qi Fang, and Jung-Woo Sohn. They provided invaluable inspiration through countless discussions and debates.

I also want to thank my previous advisor, Dr. Yih-Kuen Tsay at National Taiwan University, who played an important role when I was making my decision to go abroad for advanced study. I want to thank all my friends whose names are not listed here for their support of my work, life, and research.

This dissertation is dedicated to my family—my parents, Fang-Kuei and Ming-Hsien, my parents-in-law, Wendy and Bruce, Dr. Keng S. Liang, Dr. Yen-Hwa Hsu, Feng, and other family members. Their encouragement and emotional support made me keep moving forward. Life is a long journey and our bodies may fade away, but I believe love and spirits will last forever. Last but not least, I would like to thank my dear wife, Ting. Her wisdom and unconditional love has helped me pass through all the challenges and difficulties. I love her, and I dedicate this dissertation to her.

Chapter 1

Introduction

1.1 Research Motivation

As computer systems and networks have become more important within organizations and enterprises, side effects resulting from the growing scale of these systems have become critical issues. Contemporary computer network systems in mid-to-large size organizations tend to be very complex, consequently introducing greater challenges in their management and security control. In addition, due to the increasing demand for ubiquitous computing, more and more network-enabled devices are being developed and used in daily life, evoking an awareness that the issues we face today will likely grow more complicated in the foreseeable future.

Over time, intrusion activities affecting computer networks have occurred with increasing frequency, posing serious threats to governments, corporations, and other entities. Computer security has become a high priority, especially within organizations which deal with sensitive information. If the networked entity under attack is the military, its soldiers may be exposed to great danger. If an intrusion is targeted at acquiring business secrets, a corporation might suffer huge losses. Cyber intrusion could also introduce international tensions between countries, such that it would affect a large number of people and cause damage difficult to estimate. Clearly, cyber security and its relevant issues have become critical and are commanding increased attention. Efforts toward developing anti-intrusion methods and enhancing the roles of cyber security analysts are needed more than ever.

In order to react promptly to potential threats, it is critical to monitor organization networks and identify intrusion activities. When an incident is detected, it is highly desirable to know what kind of intrusion it is and exactly how it occurred. Understanding the process through which an attack was prosecuted can help prevent a future recurrence. For the purpose of security monitoring, several existing tools and sensors can be deployed to support event monitoring and cyber situation awareness (C-SA), a concept recently extended to the cyber security domain[1]. In addition to installing these sensors, there is a trend toward creating a cyber security situation room dedicated solely to monitoring an entire network system and to supporting an immediate response to an attack. In a cyber security situation room, however, the monitoring task usually involves a tremendous quantity of information, such that the security analysts are likely to be flooded with data. Even in network systems consisting of only a few hosts with a simple topology, the analysts may easily become confused by familiar scenarios with subtle differences. Regardless of network size, it is a challenge to deal with and react to new or unknown attack approaches. Security issues must be discovered and resolved as quickly as possible, and it is often difficult for an analyst to develop a good response under time pressure. Therefore, it is of no small importance to create computer systems to support these cyber security analysts.

1.1.1 The Missing Piece of Cyber Security Analyses

The issue of cyber intrusion detection has been studied from several perspectives [2][3]. One approach is to utilize rule-based systems to support localized intrusion detection, such as a subsystem on a particular host. Another approach relies on statistical methodologies that build a profile of normal behaviors and use it to detect abnormal behaviors. A model-based approach is built on the knowledge of how a particular attack is prosecuted. A neural network-based approach makes no assumption involving specific models or parameters, but it fails to consider an underlying causality relationship among the vulnerabilities. A common disadvantage of these approaches is that they can only deal with known incidents; for unknown attacks, time is required to identify and investigate the situation before constructing new rules or models to handle it.

The above-noted approaches are primarily used for security management on a single server. In order to obtain an overview of the vulnerabilities within an entire network, attack graphs (or vulnerability graphs) [4][5][6] have been developed and studied. An attack graph is constructed by delineating the dependencies among the vulnerabilities of an entire system. Conceptually, it can be viewed as a vulnerability roadmap toward compromising the system. An attack graph can serve as a guide for system administrators to monitor and manage a network.

Many intrusion detection systems (IDS) have been developed for detecting cyber incidents; these may be briefly categorized into host-based intrusion detection systems (HIDS) and network intrusion detection systems (NIDS). These systems can be deployed on computers or network devices to produce alerts or reports regarding known attacks based on predefined configurations. They can provide a reasonable level of information for system administrators to monitor the health of the hosts and the network.

Unfortunately, these approaches can only deal with known attacks. Therefore, human analysts are vital in new or unknown attack situations. However, the amount of data generated by these tools is usually much larger than the cognition throughput of a human analyst. Due to the confluence of huge amounts of information and the limited human cognitive load [7][8], it is challenging to ensure that a cyber analyst can keep a consistent level of concentration and make reasonable decisions under an unprecedented data supply. This challenge introduces a critical disconnection between human cognition and cyber security tools. At present, there is scant research addressing this problem, and few studies exist that show how to capture and leverage expertise to support cyber situation awareness. As shown in Figure 1.1, much research work has targeted the bottom layers of human-centric cyber security research, but the layer linking human decision support to those layers has not yet been established. Thus, a solution that can bridge this gap would make a significant contribution to the field of cyber security awareness.

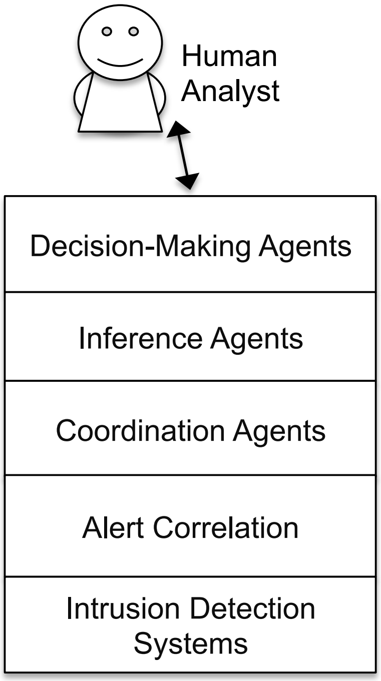


Figure 1.1. Human-centric cyber security research stack.

1.1.2 Leveraging Experts' Experience in Cyber Security Analyses

The goal of this research is to address the problem of cyber situation recognition by leveraging cyber security analysts' expertise, as a step toward developing a methodology for handling unknown or new attack approaches. Experienced cyber security experts may leverage their knowledge about attacking patterns to recognize and identify potential incidents according to reported alerts. However, intruders may act at any time from anywhere around the world, such that monitoring cyber events becomes a heavily loaded task. Without guidance from expert experience and knowledge, there is potential for a combinatorial explosion in the search space of identifying and analyzing undesirable incidents. Consequently, it would be useful to develop a methodology for building systems with the capacity to retain experts' knowledge, to share their workload, and to cover their blind spots.

Humans are capable of learning from history, and experience can be accumulated that enables us to prepare for similar situations in the future. Experience plays an important role: what we learn from past incidents can help us focus on present important and specific conditions at times when it is difficult to investigate and review all possible influencing factors. Research on leveraging human experience had been studied for different purposes and across various domains [9]. Instance-based learning theory (IBLT) [10] was proposed to explain the process of how a naïve becomes an expert through experience accumulation; this theory has been proven through simulations using the ACT-R cognitive architecture [11]. Experience-based decision making has also been studied when overlaid on the R-CAST agent architecture [12][13], a multi-agent framework featuring variations of the recognition-primed decision (RPD) model [14][15]. In addition, experience matching under a large feature space has been studied and addressed using geometric diffusions [16]. For cyber situation awareness, experience-based approaches are also emerging which have captured research attention [17].

Experience-based recognition and matching is a promising problem-solving approach, but the details of solutions can vary from one particular domain to another. It requires the knowledge of a specific domain in order to decide how to capture a current situation, how to represent experience and corresponding knowledge, and how to match experience to that current situation. Therefore, in order to develop an experience-based approach for bridging the gap between human cognition and cyber security tools, the characteristics of a network system from cyber security's perspective should be well identified.

1.2 Problem Description

In order to apply experience-based cyber security analysis, it is fundamental to capture the characteristics of a network system from the perspective of cyber intrusion detection.

First, a network system has vast quantities of information or events continuously happening within it. Multitudes of network activities, either normal or malicious, run throughout the entire system. A cyber security analyst needs to identify potential attacks based on an enormous number of activities or events.

Second, there are causal relationships among a network's events due to the dependencies among the vulnerabilities, where one or more events serve as the precondition for another event. For example, if a target host is protected behind a server, the attacker needs to compromise this server before reaching the target. The action of compromising the server is a precondition for attacking the target host. Causal relationships can vary in different parts of the system according to the vulnerabilities depicted in the attack graph [4][6][18]. Some events may have only one precondition, while others may have multiple conjunctive preconditions.

Third, the event of interest or significant importance can be implicit or hidden. Collecting information is critical to making a good decision. However, this is not always a straightforward task because some information may exist implicitly or be hidden behind other factors, and some information may be inaccessible or very difficult to obtain. Therefore, additional sensors or probes may need to be allocated in order to obtain the information. For instance, file integrity checking tools such as Tripwire [19] may be used to check whether the file system on a specific host has been modified, but additional time and cost are required to make the tool available to the analyst.

Fourth, the task of an analyst is to continuously monitor a system and determine whether there is any undesirable situation occurring in it. For example, one duty of a corporate information technology service department is to monitor abnormal events that may indicate potential attack activities against the corporation's network. The task is to decide whether there is an attack, what type of attack it is, and how the attack is being prosecuted.

These four characteristics are of significance when developing experience-based cyber security analytics. They can be used to form the basis for building an abstract model to guide the way in which an experience can be constructed and refined.

In this research, we address the decision making problem under the cyber security problem domain, which consists of events that are difficult to observe by analysts but may be detected indirectly by sensors. Specifically, our goal is to identify potential incidents such that an undesirable incident can be controlled before it develops into a catastrophic event. The characteristics of this problem can be summarized as follows:

- Intrusion incidents are made by malicious entities, which tend to approach or attack the system through an abnormal and unpredictable way. Each intrusion incident may consist of several steps, where each step is a discrete attack against the system.

- In order to obtain an overview of all possible types of intrusions, all the known vulnerabilities can be integrated into an attack graph with their dependency relationships [20]. An attack graph captures the dependency relationships among vulnerabilities but does not deal with concrete alert instances. It is a roadmap showing all possible routes toward compromising a system.

- Given an attack graph, it is essential to identify important attack steps and formulate them as types of events. Since the event types come from the attack graph, there are known causality relationships among them. These come from the dependencies among the vulnerabilities existing in the system. An attack step cannot happen if its precedent attack step does not succeed.

- The events of interest cannot be directly observed, but they can be indirectly monitored through alerts generated by sensors or detectors. These alerts form a sequence and are sent to the decision maker. Each alert is associated with a hidden event and thus contains the information about both the alert itself and its associated hidden event.

- An event can trigger multiple alerts of different types, if more than one sensor is installed and associated with this type of event.

- In an ideal case, every sensor is deployed for monitoring exactly one type of event. However, due to budget constraints or other technical considerations, a sensor may be installed for monitoring a group of hosts or more than one type of event; therefore, it might sometimes be difficult to retrace an exact type of event given an alert generated by this kind of sensor.

- An alert may be delayed.

- An alert may be missing.

- Multiple incidents could occur in an interleaved manner.

- There exist alert correlation engines that process alerts and generate correlation reports. These reports indicate which alerts are correlated [21]. An alert correlation engine can indicate which alerts are correlated but cannot guarantee which alerts are not correlated.

- In addition to the attack graph, an alert correlation report can also serve as a clue to the causality relationship among the underlying events indicated in the report.
- A cyber security analyst is asked to make decisions based on all observable information from a system, including both run-time alert sequences and run-time correlation information. In order to make a decision, the following criteria are important and should be under the analysts' consideration:
  - Causal relationships among the hidden events
  - Temporal order of the hidden events
  - Temporal order of the observable alerts
  - Correlation information

In this research, we will address and answer the following research questions:

- How can we reduce the cognitive load of a cyber security analyst?
- How can we formalize the structure of a network system from the viewpoint of cyber intrusion detection?
- There is a combinatorial explosion in the search space of identifying and analyzing undesirable incidents from the run-time information. Leveraging experts' experience in cyber intrusion detection would reduce the search space. How can we capture and leverage experts' experience to make decisions?

1.3 Research Scope

The aim of this research is to address the problem of cyber situation recognition from the viewpoint of leveraging cyber experts' experiences and reflections. The research scope covers both the field of cyber security and the field of decision making. More specifically, naturalistic decision making based on human experience [15] is a particular field that is relevant to our study.

1.4 Dissertation Outline

The remainder of this dissertation is organized as follows. Chapter 2 outlines the relevant research background and related work. Chapter 3 gives the details of the experience-based approach for cyber security analysis, including the specification of the partially observable event-alert model and the concept of experience relaxation. The engineering and implementation of a decision support system based on the experience-based approach is described in Chapter 4, followed by simulations and experiments in Chapter 5. Chapter 6 concludes the dissertation.

Chapter 2

Background and Related Work

This overview of the background and related work first addresses the literature regarding network security, and then moves on to decision making and decision analyses. Next we discuss the concepts of data fusion and situation awareness as well as that of an information supply chain, since information is a key factor to the success of situation awareness and decision making. Finally, because the network vulnerabilities and their dependencies can be extracted as a graph model featuring heterogeneous types of nodes and links, we end this chapter with a review of network science literature.

2.1 Network Security

Computer network security has been studied from multiple perspectives using various approaches [2][3]. Over time, assorted types of intrusion detection systems (IDS) have been developed. Some intrusion detection systems are designed to identify incidents based on known attack patterns, while others function by making profiles of normal behaviors and identifying anomalies based on them [21].

As noted in Chapter 1, intrusion detection systems can be categorized into host-based intrusion detection systems (HIDS) and network intrusion detection systems (NIDS). Both types can be deployed to generate an alert or an incident report based on predefined configurations. An HIDS is an application deployed on a host which detects intrusions based on logs, specific files, system invocations, or other internal behaviors. OSSEC [22] is an example of a host-based IDS. A

NIDS does its work by monitoring network traffic and examining the content of exchanged packets. Snort [23] is an example of a network IDS presently popular in the industry. To augment these systems, applications can be developed and overlaid to satisfy more specific detection requirements or goals. For example, the work of [24] is based on the use of a nondeterministic automaton to analyze the equivalence of rules on individual hosts, an idea evaluated using Snort.

In addition to intrusion detection systems, anti-malware applications also play an important role in network security. These are designed to detect the installation or execution of malicious software such as viruses, worms, or Trojan horses [25]. Another intrusion detection tool is a file integrity checker, designed to monitor modification activities on a file system such as the creation or deletion of files. An example of such type of system is Tripwire [19].

With assistance from these tools, intrusions and incidents occuring in local portions of a network may potentially be detected. However, given the enormous volume of data generated by these tools, it has become highly desirable to use an alert correlation mechanism that can filter out noise and process scattered alerts to generate more meaningful information.

Alert correlation is a key part of cyber security research. It has been studied and addressed using various approaches and techniques [21][26]. Its fundamental work is alert aggregation. To perform this, a prerequisite step is to convert alerts having different layouts into a consistent format. The normalization of alerts can be realized through the use of a common encoding language such as the Intrusion Detection Message Exchange Format (IDMEF) [27], where the messages in IDMEF can be exchanged with the Intrusion Detection Exchange Protocol (IDXP) [28][29]. The normalization of alerts can also be achieved by referring to public vulnerability databases such as Common Vulnerabilities and Exposures (CVE) [30] or BugTraq IDs [31].

Alert aggregation may be based on the similarity in attributes such as attacker's IP address, target's IP address, port number, attacking time, or class of attack [32][33]. An alternative approach to alert aggregation is clustering, which uses dissimilarity in attributes as the metric for determining the distance between each pair of alerts [34]. While addressing the alert correlation problem in a systematic and flexible way, the clustering approach is less persuasive when the cluster size and the weight of each attribute are difficult to determine.

Other methodologies using machine learning techniques based on their similarities have been developed [35][36]. The causality relationships among alerts are also addressed in certain alert correlation research [21]. Scenario-based or state-based correlation tests whether or not a set of alerts can be assembled as an attack scenario [37][38]. This approach requires a preknowledge of possible attack scenarios, which can be described in formal specifications such as STALL [39] and LAMBDA [40]. Rule-based correlation is performed by checking and matching the preconditions and postconditions of the alerts [41][42][43][44][45][46]. As for statistical correlation, the approaches in [47] and [48] leverage Bayesian networks to statistically construct causal relationships among the alerts; however, obtaining a precise conditional probability table (CPT) may be challenging work when using the Bayesian approach.

As for causality relationships, it is feasible to obtain a comprehensive list of known vulnerabilities and their dependencies from a given network system. These vulnerabilities can then be integrated into a dependency graph, also called an attack graph or a vulnerability graph [5][49]. An attack graph can serve as a fundamental guide for recognizing possible attacking steps under a particular network system.

Given a network configuration including all networking devices, firewall rules, operating systems and software running on the hosts, as well as other relevant settings, an attack graph can be constructed to show all possible intrusion paths towards compromising a given system [4][6][18][20]. The graph is a deterministic diagram that can be automatically generated using software tools such as CAULDRON [50].

Although attack graphs appear in different forms in the literature, their concept and purpose are the same. A graph consists of a number of vulnerabilities with their preconditions and postconditions. Each node in an attack graph represents a single step, which can be either the compromise of a service on a specific machine or some form of state change on a host. The edges in an attack graph show the causality relationships between the nodes, and the precedent node is always the precondition of its subsequent node. A node becomes reachable only after all its preconditions are satisfied. Based on the graph's information, a cyber security expert can use it as guidance to identify potential attacks and to take corresponding actions for handling the situation. Moreover, an attack graph can serve as the basis for building other security analysis tools; for example, the Bayesian network approach in [51] builds networks based on underlying attack graphs.

There is plenty of research addressing elements of network security. However, little work exists connecting cyber security analyst expertise with cyber situation recognition. The aim of this research is to bridge this gap from the perspective of leveraging human cyber security experts' experiences, which is directly related to the area of decision making. Therefore, the following section provides an overview of research on decision making and decision analysis.

2.2 Decision Making and Decision Analysis

Decision making is a long-standing challenge, ubiquitous across various domains such as business strategizing, financial planning, situation awareness for battlefields, anti-terrorist agency policy making, and cyber situation awareness [1]. Although these domains have different emphases and operate at different scales, they share common properties; thus they should be able to be addressed in a general way, allowing for specialized designs in the details.

In general, decision making is the process of situation evaluation involving information collection and human cognitive activities in order to produce final choices from among alternatives [52]. Decision making involves multiple cognitive steps and activities. How humans make decisions has been studied for decades, and relevant models exist in the literature, including the utility theory [53][54], ideal rational decision-making [55], and naturalistic decision-making [14][15]. To evaluate the situation for a complex problem domain, decision makers not only need to process information but also to infer facts lying beneath the surface. In this circumstance, decision makers may easily be distracted or overloaded by multiple, simultaneous activities. Therefore, it may be beneficial to use a computer-based decision support system to assist in analyzing those complex problem elements which are difficult for humans to clarify but relatively easier for computers.

Historically, relevant decision-making research can be traced back as far as the 18th century to Bernoulli's measurement of risk (1738) and Bayes' theorem

(1763). More recently, brief history of decision analysis research across economics, statistics, and psychology was provided [56] by Smith and von Winterfeldt. These authors identified three different perspectives in decision-making research: 1)the normative perspective, focusing on rationality and normative models as the guidance for making a decision; 2) the descriptive perspective, focused on how humans actually make decisions, and what they think and how they behave; and 3) the prescriptive perspective, for improving decision making by "using normative models, but with awareness of the limitations and descriptive realities of human judgment." (p.562) Smith and von Winterfeldt showed how prescriptive decision analysis research is established on the basis of normative and descriptive decision making.

The challenge of decision making has been studied from multiple perspectives. Significant research from the psychological perspective been conducted to analyze how people make decisions. Studies have shown that the way humans actually make decisions is quite different from the ideal "rational model." According to prescriptive models, a rational decision-making process consists of the following steps: defining the problem, identifying criteria, weighing the criteria, generating alternatives, assessing each alternative on each criterion, and accurately calculating and selecting the optimal decision [55]. However, studies from the descriptive viewpoint have demonstrated that human rationality is bounded such that decision making is usually performed according to specific heuristics (or selection strategies) such as availability, representativeness, anchoring and adjustment, and other strategies [57]. With bounded rationality, humans are also likely to be affected by bias—another research element within the psychological realm.

In 1989, the recognition-primed decision (RPD) model [14][15] was proposed to explain how humans make decisions under time-stressed situations. It was found that rather than trying to choose the optimal option, human experts tend to react quickly to limited information by iteratively evaluating available alternatives and selecting the first satisficing one. A later study focused on instance-based learning drew a consistent conclusion [10]. In it, an instance-based learning mechanism was implemented on top of an ACT-R cognitive architecture [11], and the learning curves of both naïve and expert decision makers (produced by simulations) confirmed the instance-based learning theory (IBLT) [10].

A common goal of decision-making research is to improve the quality—usually defined as the outcome—of the decision, which can be evaluated based on its correctness. The correctness of decisions is also influenced by the biases introduced as a result of human cognitive flaw or incomplete information [55].

In addition to individual decision making, human behavior in group decision making has been well studied. In group decision making, the quality of decisions is affected by a number of factors such as information sharing, group norms [58], member familiarity and information distribution [59], and decision biases such as groupthink [60]. Moreover, previous studies have shown that the decision-making process itself is also an important factor relevant to the outcome of the decision [61].

Research from the psychological perspective has provided insight as to what decision-making is, thus forming the basis of the design of computational models for decision entities. This is especially helpful for developing decision support systems involving human-machine interactions.

From the artificial intelligence perspective, decision-making problems have been studied in several ways. One approach has been to simulate individual human decision-making activities using a computational model. Through observation and analysis of how humans make decisions, a psychological decision model can be transformed into a computational model with well-defined logic and algorithms. Due to the restrictions of computer systems and the differences between humans and machines, a psychological decision model can be realized in different ways. For example, the RPD model [15] has been realized in the RPD-Enabled Collaborative Agents for Simulating Teamwork (R-CAST) multi-agent architecture in three different ways [12][13], where each is based on experiences organized into a tree-like structure. In contrast, the RPD model has another implementation using experiences organized in a flat structure [62].

Using a computational model, knowledge engineers can work with domain experts to build an intelligent system. This in turn can be utilized to build decision support systems.

Decision support systems have been studied and developed using various approaches. One of the traditional forms is the expert system [63], typically equipped with a question-and-answer interface. The underlying implementation of a decision support system can be a knowledge base with inference capabilities [64], information retrieval techniques, data mining methodologies, or other computationally feasible approaches. One of its advantages is the ability to reduce the cognitive load of human decision makers by providing more organized and better-grained information, perhaps even with friendly human-computer interfaces.

If we consider a decision-making task that involves a group of people, not only the cognitive status of each individual decision maker but also his or her interactions with other group members will play important roles in the process. Group decision-making is performed by collaborative entities. In this circumstance, the group of entities can be regarded as a team in which all the individuals work cooperatively to make decisions [65]. As with group decision making, a decision support system can engage a group of people. Therefore, the focus of a group decision support system is not limited to individual decision makers; instead, the whole decision process is incorporated into the system to support the cooperative style of group decision-making.

The Decision Process Model [66] is a concept capable of representing a specific decision-making process and its internal activities. It is a hierarchical model consisting of multiple expandable sub-processes, all of which can be stated in process definition languages such as Business Process Modeling Notation (BPMN) [67]. Many decision-making activities are cooperatively made by a group of people using a predefinable decision process in order to minimize the risk and pursue a better outcome. For example, Figure 2.1 illustrates a military decision process involving multiple decision steps for dealing with potential targets. The process is predefined based on a standard operating procedure. Each rectangle in the figure represents a single step in the decision process, and the sequential relationships are indicated with arrows.

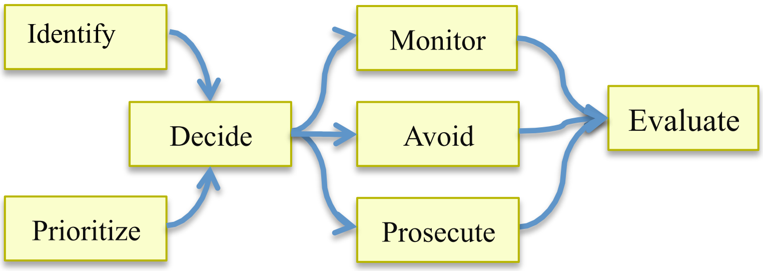


Figure 2.1. An example decision process from the military domain.

Another tool related to the Decision Process Model is the influence diagram, also known as a decision network [68], proposed by Howard and Matheson. Influence diagrams have been adopted by the artificial intelligence community as a method for modeling decision making under uncertainty [69]. An influence diagram contains three types of nodes: chance nodes represent random variables, decision nodes represent the points where decision makers need to make a decision, and utility nodes are function locations for calculating final utility based on the input from other nodes. Howard and Matheson also provided a method for solving a decision network by deriving a decision tree from it, but this approach can be computationally costly since the tree may be exponentially large. In order to evaluate influence diagrams, Shachter provided a solution that directly solves a decision network without an intermediate decision tree [70]. Later, Zhang et al. provided additional insight into how to evaluate decision networks from a computational viewpoint [71]. They introduced the concept of decomposability to decision networks and introduced stepwise-decomposable decision networks (SDDN). An algorithm based on a divide-and-conquer strategy was also developed for evaluating an SDDN. Since their development in the mid-1970s, influence diagrams have been used and applied in diverse areas [72][73], including the analysis and evaluation of team-based decision making under uncertainty by Detwarasiti and Shachter [74]. However, the use and application of influence diagrams in some fields such as medical decision analysis still remains limited [75].

2.3 Data Fusion and Situation Awareness

As articulated by Hall and Llinas, the intent of data fusion is to combine and integrate data from multiple sensors to achieve a higher level of understanding of the entity of interest in the environment [76]. Data fusion can take place at different levels. At the level of raw data fusion, raw observational data are combined. At feature-level fusion, representative features are extracted from the data. At decision-level fusion, sensor information is further processed using methods such as weighted decision techniques, Bayesian inference, or the Dempster-Shafer theory. To standardize communication among researchers and developers, the terminology related to data fusion was defined by the Joint Directors of Laboratories (JDL)

Data Fusion Working Group, and the JDL Data Fusion Process Model was developed. The JDL model is a framework for data fusion systems, linking information sources and human-computer interactions through a meshing of processing at level 1 (object refinement), level 2 (situation refinement), level 3 (threat refinement), and level 4 (process refinement), and database management systems.

Based on the JDL model, Steinberg et al. proposed revisions and expansions to facilitate the development and operation of multi-sensor systems [77]. These included broadening the model beyond its original military focus and refining its taxonomy. Level 0 (sub-object assessment) was added to incorporate data association and characterization at the pixel or signal level. Llinas et al. proposed another expansion to the JDL model [78]. This work was focused on better understanding the internal processing in a fusion node, and incorporated extensions to the model such as quality control, reliability, and the consideration of distributed data fusion (DDF). Later, Llinas et al. proposed another expansion to the JDL data fusion model [78]. Their work was focused on better understanding of internal processing in a fusion node and some extensions to the model such as quality control, reliability, and the consideration of distributed data fusion (DDF).

Tangentially, information is a key factor to the success of decision-making. Accurate information can lead to better evaluation of a situation. Information may be collected from multiple sources and might need to be consolidated and/or further processed. These procedures may be viewed as value-added activities, where each single step provides an improvement to the quality of the information.

The concept of the value chain as proposed by Porter [79] is a linkage of value-adding activities. A product passingthrough the chain can gain value from each activity, such that the overall value produced by the chain is more than the sum of the value added by each individual activity. This concept can be further applied to the entire supply chain to include value added from other cooperative organizations. The object of value-adding in the value chain is not limited to products; information can be improved through a value chain as well.

The concept of an information supply chain (ISC) proposed by Sun and Yen [80] was developed to address information sharing issues in a teamwork environment. To achieve efficient and effective teamwork, each team member is expected to anticipate the information needs of other teammates and proactively provide information to satisfy those information requirements. Sun and Yen's research covers information anticipation using a cognitive decision model, information requirement planning (including requirement consolidation), and the integration of multiple information acquisition strategies.

Surana et al. suggested that rather than as a literal chain, the concept of a supply chain should be considered a network [81]. Further, Choi et al. [82] and Pathak et al. [83] brought the concept of complex system or complex network into supply networks. They suggested that the idea of complex system or complex network should be incorporated in the design of a supply chain.

As a network, additional supply channels can be established in a system such that its robustness can be improved. Thadakamaila et al. pointed out that the topology of a supply network can affect its fault tolerance capability [84]. On the other hand, it was also noted in Rice Jr. and Caniato's research [85] that in a supply network, a small portion of failure would possibly crash the whole system.

2.4 Network Science

Since the vulnerabilities and their dependencies within a network can be extracted as a graph model with heterogeneous types of nodes and links, it is desirable to explore relevant research in network science.

Network science has emerged as a research area which draws attention and interest from people across a wide range of backgrounds. In 1998, Watts and Strogatz proposed the model of small-world networks [86] in which they demonstrated that the distance between two arbitrarily selected nodes in a social network is surprisingly short, in terms of number of edges. The model of scale-free networks proposed by Barabási and Albert shows a similar property [87]. The network generation algorithms of both models are regarded as capable of creating networks similar to those in the real world.

Barabási and Albert investigated network topologies and connectivities from multiple domains and determined the properties of scale-free networks as well as how they are constructed [87][88]. A scale-free network is one in which the connectivities of the nodes follow a power law distribution. This type of network is ubiquitous in many fields where networks exist such as physics, geography, biology, sociology, and computer science. Evidence shows that a scale-free network is formed by the process of continuously adding a new node to an existing network, preferably a higher degree node. This observation provides insight into complex systems from a network's point of view.

Newman reviewed the developments in the field of complex networks [89], including random graphs, the small-world model by Watts and Strogatz [86], degree distribution, network correlations, clustering, the models of network growth, and the behavior of dynamic processes that take place in networks. Boccaletti et al. published a survey on elements of complex networks such as network structure, network robustness, and cellular automata on topologies [90]. They also addressed algorithms for finding community structure, navigation and searching within a network, and the modeling of adaptive networks. Costa et al. published a survey specifically on the measurements of complex networks [91]. In it, main existing measurements of networks were addressed; in addition to simple features such as node degree, shortest path length, and the clustering coefficient, more powerful measurements from a topological perspective (such as connectivity) were also explored.

As for the topic of network topologies and their robustness, Albert et al.'s research revealed why many complex systems in the world have a strong error tolerant capability [92]. They demonstrated that the scale-free network is significantly tolerant of random failures, thus explaining why many existing complex systems having this scale-free property can continue to function when portions of their components become unavailable or are in error. Logically, however, scale-free networks are still subject to failure if their most highly connected nodes are malfunctioning or under attack.

Holme et al. conducted a study on complex networks subjected to different attack strategies on nodes and edges; their results provide a guideline for how to protect a complex network according to a topological perspective [93]. After implementing the attack strategies, they measured and compared the damage. The attack strategies included the removal of the node/edge with 1) the highest initial degree, 2) the highest initial betweeness, 3) the highest recalculated degree, and 4) the highest recalculated betweeness. To measure the damage caused by the attacks, two kinds of metrics were defined: the average inverse geodesic length and the size of the giant component. They applied the experiments to two real networks and four theoretical networks: random, small-world, scale-free, and clustered scale-free. The strategies based on recalculated information were shown to be more efficient than those based on initial information, thus implying that it is important to consider the change in a network structure during an attack process in order to efficiently protect a network. Significantly, their results also indicated that none of the four theoretical models resembled the two real networks, which means there is ample room for improving the theoretical models. The results also showed that the random graph model is more robust than the other subject networks, implying that a serverless network would be more robust to attack.

According to Newman, the assortativity of a network can affect its robustness [94]. Assortative mixing is a network property in which high-degree nodes tend to attach to other high-degree nodes. Newman found that assortative networks are more robust than disassortive networks to the removal of high-degree nodes. His explanation is that these high-degree nodes tend to be clustered in the core of the network, such that their roles in the network are somewhat redundant. In this study, assortativity coefficient is defined for measuring the assortativity of a network. The measurement was taken on social networks, technological networks, biological networks, and three other specific kinds of networks including random graphs [95], the growth graph model of Callaway et al. [96], and Barabási and Albert's growth model [87]. However, the result of Barabási and Albert's growth model is inconsistent with the expected outcome, thus indicating the incompleteness of the metric.

In 2008, Grubesic et al. published a paper providing a review of the approaches to assess network robustness and vulnerability to disruption or interdiction [97]. These approaches can be categorized into scenario-specific, strategy-specific, and structured approaches. The article include an empirical study based on the Abilene network. Both global metrics and local graph theoretic measures were taken from the network and showed different results, leading the authors to conclude that the analyses for both scales need to be considered together. In addition, connectivity and network flow vulnerability were also assessed under seven cases, including the best-case and the worst-case scenarios that form the upper and lower bounds for disruption impacts.

The percolation of networks is also of interest as it relates to cascading node failures in networks. Callaway et al. present a solution for predicting the behavior of networks under cascading breakdowns [98]. They used generating function methods to study the behavior of various percolation models on random graphs with different node degree distributions. The results show that networks following a power-law node degree distribution are robust against the random removal of nodes but fragile in response to the removal of highly connected nodes.

Kong and Yeh presented a study of the cascading node failure problem in wireless networks from a percolation-based perspective [99]. Using simulation to test their predictions, they found that cascading node failure in large-scale wireless networks can be modeled as a degree-dependent site percolation process on random graphs.

Network resilience and efficiency are two further dimensions for evaluating a dynamic network, and there are trade-offs between the pursuit of either one. Gutfraind presented an approach to achieve an optimal network by striking the balance between resilience and efficiency [100]. It involved formulating the constraints as a multi-objective optimization problem based on the metrics defined for efficiency and resilience.

On a final note, Carley et al. presented a dynamic network analysis tool kit developed at Carnegie Mellon's CASOS laboratory [101]. This tool kit is capable of handling network data with a mixture of node types (multi-mode) or various types of relations between any two nodes (multi-plex), as well as networks with data changing over time. It also provides functionalities such as data extraction, visualization, analysis, and simulation. For data persistence, this tool kit supports the exchangeable data format DyNetML, which was proposed by Tsvetovat et al. [102].

Chapter 3

Experience-Based Analytics for Cyber Situation Recognition

In order to address the challenge of cyber situation recognition, we developed an experience-based approach inspired by Klein's recognition-primed decision model [15]. We also constructed the framework for a partially observable event-alert system to use as a ground model to support analysts' decision-making. Then, in order to analyze how human experts make decisions about similar situations based on the experience of discrete past incidents, we introduce the concept of experience relaxation, intended to utilize a limited number of experiences by extending their applicability.

In this chapter, we first investigate the concept of experience and how it can help in decision making. Second, we generate a formal specification of the partially observable event-alert system, which serves as the basis for assisting decision making and experience relaxation in cyber security analyses. Third, we discuss how experience can be captured, represented, and used for supporting cyber situation recognition. Finally, the concept of experience relaxation with its logical foundation is introduced.

3.1 Experience-Based Situation Recognition and Decision Making

According to psychological studies, experience plays a very important role in human decision making. The recognition-primed decision (RPD) model [15] reveals how decision making under time stress is performed by cue matching against past experiences. Coincidentally, a study from a different point of view, instance-based learning theory [10], draws a consistent conclusion, proving that experience is accumulated through the learning process as one progresses from naïve to expert.

As mentioned in 2.2, one approach to studying decision making based on the artificial intelligence perspective is to simulate human decision-making activities with computational models. By applying this method, a psychological decision model can be transformed into a computational model with well-defined logic and algorithms. In it, experience and knowledge elicited from domain experts can be used to build experience bases, which may then be used in corresponding computer systems.

The idea of the experience-based approach is conceptually depicted in Figure 3.1. Experience accumulation and situation recognition are two major components of this process. Past intrusion incidents, intrusion reports, and relevant analyses can be processed and used as an experience base. When a new alert sequence takes place, an analyst can match this sequence with data in the experience base to find similar situations from the past, a process wherein each situation can be retrieved and presented with its associated reports and/or reflection letters from previous analyses. The recognition results can help the analyst to make decisions and take corresponding actions. After the intrusion is properly handled, the new decision and the analyst's reflection can also be captured as new elements of the experience base. These steps may recur in turns and form a continual process; in the meantime, the experience base continues to expand and contains more information for supporting future recognition.

The experience-based approach is intended to solve the decision-making problem in a way that not only can incorporate domain experts' experience for known attacks but also can quickly capture new types of attacks in terms of their signatures. Additionally, it is desirable to have the experience construction process

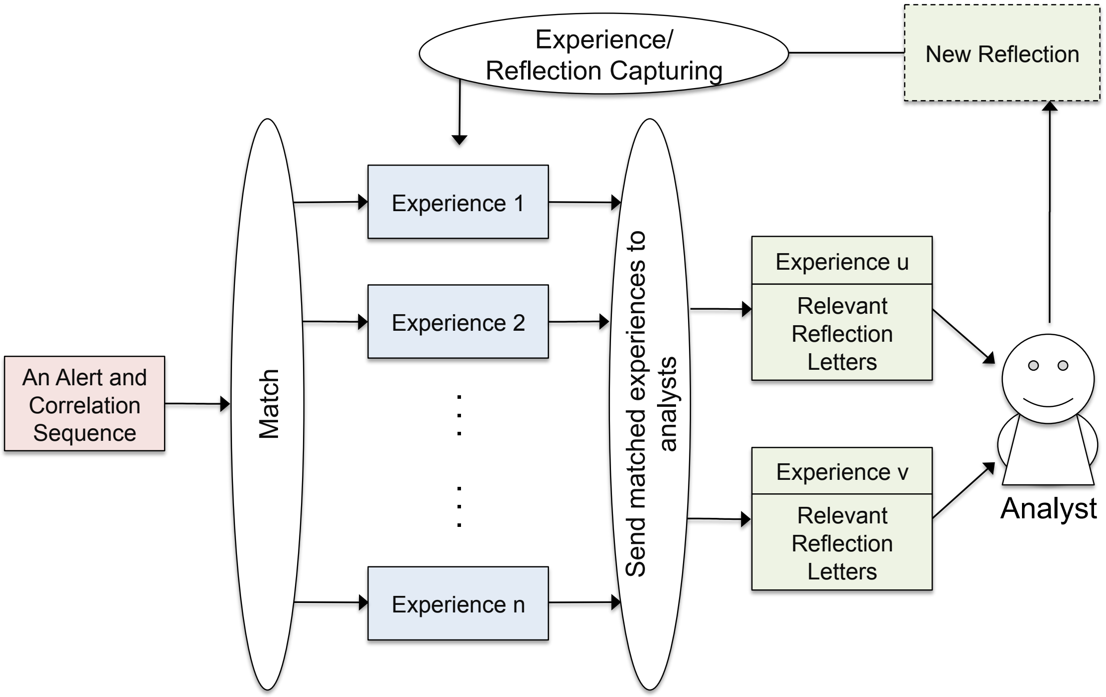

Experience/
Reflection Capturing
New Reflection
Match
Send matched experiences to analysts
An Alert and Correlation Sequence
Experience 1
Experience 2
Experience n
Experience u
Relevant Reflection Letters
Experience v
Relevant Reflection Letters
Analyst

Figure 3.1. Experience-based situation recognition.

automatically performed by the system. We believe that the creation and nurture of a knowledge container of experiences is the key to building the bridge between cyber security situation awareness and analysts' knowledge.

Another benefit of leveraging experiences for decision making is that the search space can be reduced by checking only the experience extracted from domain experts instead of reviewing all situations, many of which are not likely to happen. Instead of assessing all possible combinations, attention is paid to what has been captured in the experience base. In this way, the cognition overhead of decision makers can be reduced.

3.2 Partially Observable Event-Alert System

In order to deal with the problem, we set out to develop a formal model to capture the characteristics of a network system from the viewpoint of intrusion detection. In this section, we identify some important system features using an example, and then present a formal model specification of the partially observable event-alert system. This model provides an abstract view of a complex system in terms of the events of interest, their causality relationships, and their corresponding observable alerts. This model can serve as a basis for assisting decision making and experience relaxation for cyber security analyses.

3.2.1 Important Features for Cyber Security Analysis

Given a network system, an attack graph can be constructed to show its vulnerabilities and their dependencies. For intrusion detection, particular types of events of interest in the network can be identified based on the guidance from its corresponding attack graph. Since electronic events in a computer system cannot be seen by human beings, in order to monitor these events of interest, sensors such as HIDS, NIDS, anti-malware, or other devices must be deployed by the administrator. These sensors will generate alerts when the driving events occur.

For example, the scenario from [49] contains three hosts in a simple topology. This configuration allows two possible ways of exploits as shown in Figure 3.2.

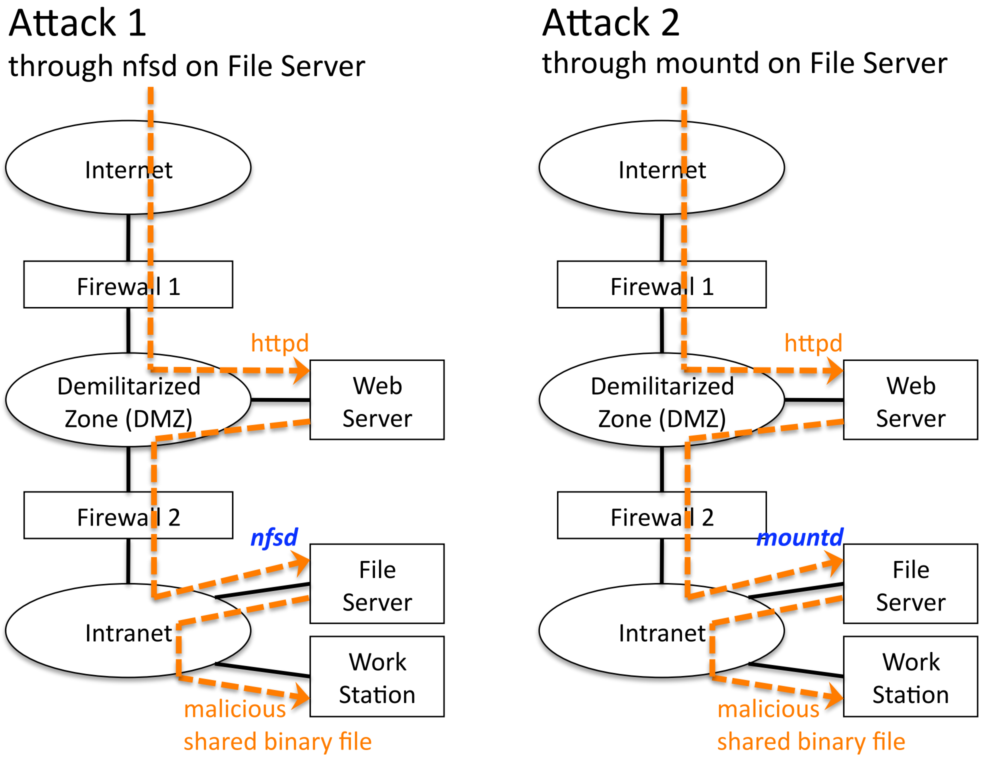


Figure 3.2. Attack scenarios on the example network.

Based on the attack graph for this scenario from [51], six types of events and eight corresponding types of alerts can be identified as shown in Figure 3.3, where each dashed arrow indicates the triggering relationship from an event to an alert.

From the diagram in Figure 3.3, additional features can be extracted as follows. First, there is a list of alert types as shown in Figure 3.4, which are observable to the analyst. Second, each alert contains information about its triggering event. This relationship is indicated by the dotted lines running left to right in Figure 3.5. The events are often hidden from the analyst (i.e., not directly shown to the analyst), and the analyst often uses alert reports to identify interesting events and then look into them. Third, according to the attack graph, events are actually linked by their causal relationships as indicated by the solid straight lines in Figure 3.6. Fourth, these causal relationships imply a typical temporal order of alerts, as

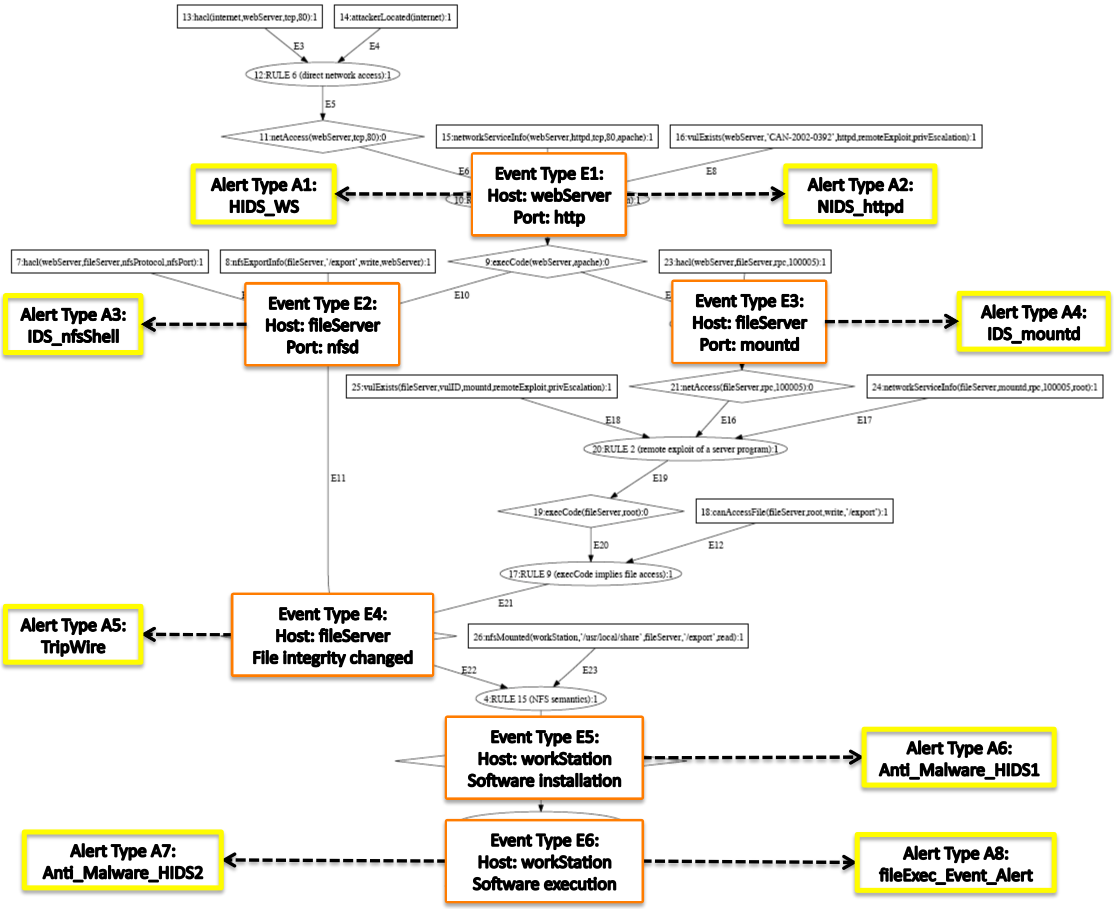

13:hacl(internet,webServer,tcp,80):1
14:attackerLocated(internet):1
E3
E4
12:RULE 6 (direct network access):1
E5
11:netAccess(webServer,tcp,80):0
15:networkServiceInfo(webServer,httpd,tcp,80,apache):1
16:vulExists(webServer,'CAN-2002-0392',httpd,remoteExploit,privEscalation):1
E6
E8
Alert Type A1: HIDS_WS
Event Type E1: Host: webServer Port: http
Alert Type A2: NIDS_httpd
7:hacl(webServer,fileServer,nfsProtocol,nfsPort):1
8:nfsExportInfo(fileServer,'/export',write,webServer):1
9:execCode(webServer,apache):0
23:hacl(webServer,fileServer,rpc,100005):1
E10
Alert Type A3: IDS_nfsShell
Event Type E2: Host: fileServer Port: nfsd
Event Type E3: Host: fileServer Port: mountd
Alert Type A4: IDS_mountd
25:vulExists(fileServer,vulID,mountd,remoteExploit,privEscalation):1
21:netAccess(fileServer,rpc,100005):0
24:networkServiceInfo(fileServer,mountd,rpc,100005,root):1
E18
E16
E17
20:RULE 2 (remote exploit of a server program):1
E11
E19
19:execCode(fileServer,root):0
18:canAccessFile(fileServer,root,write,'/export'):1
E20
E12
17:RULE 9 (execCode implies file access):1
E21
Alert Type A5: TripWire
Event Type E4: Host: fileServer File integrity changed
26:nfsMounted(workStation,'/usr/local/share',fileServer,'/export',read):1
E22
E23
4:RULE 15 (NFS semantics):1
Event Type E5: Host: workStation Software installation
Alert Type A6: Anti_Malware_HIDS1
Alert Type A7: Anti_Malware_HIDS2
Event Type E6: Host: workStation Software execution
Alert Type A8: fileExec_Event_Alert

Figure 3.3. An attack graph with events of interest.

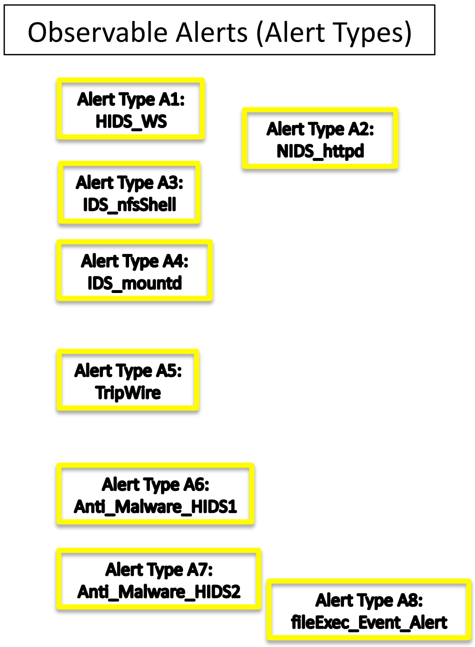

Observable Alerts (Alert Types)
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A3: IDS_nfsShell
Alert Type A4: IDS_mountd
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A7: Anti_Malware_HIDS2
Alert Type A8: fileExec_Event_Alert

Figure 3.4. Observable alerts.

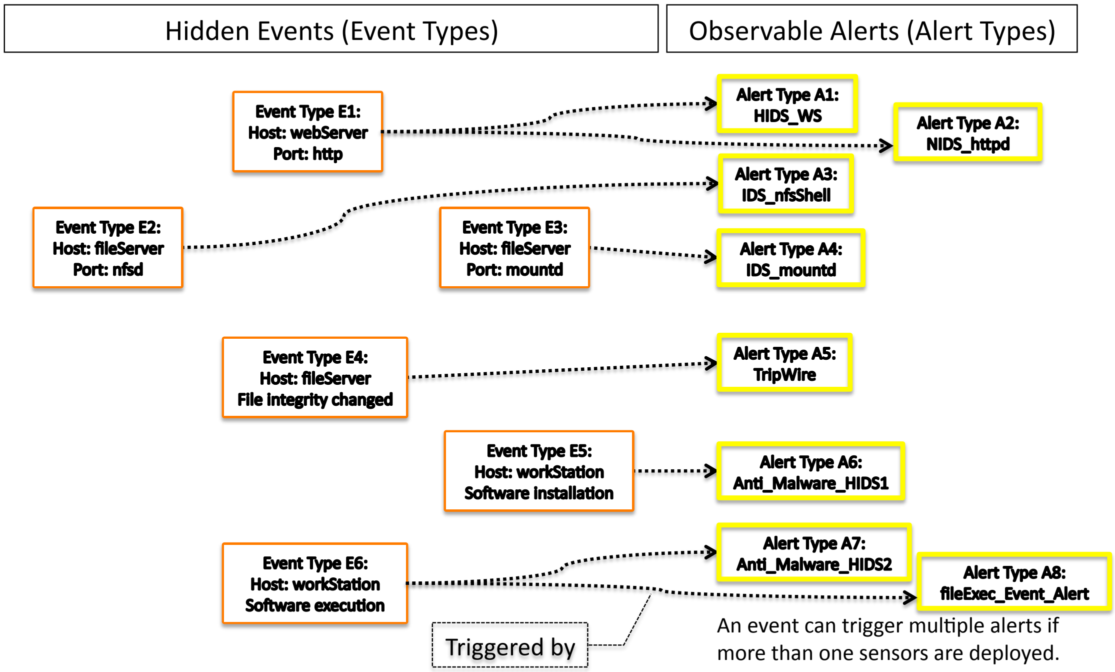

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E2: Host: fileServer Port: nfsd
Event Type E3: Host: fileServer Port: mountd
Event Type E4: Host: fileServer File integrity changed
Event Type E5: Host: workStation Software installation
Event Type E6: Host: workStation Software execution
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A3: IDS_nfsShell
Alert Type A4: IDS_mountd
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A7: Anti_Malware_HIDS2
Alert Type A8: fileExec_Event_Alert
Triggered by
An event can trigger multiple alerts if more than one sensors are deployed.

Figure 3.5. Each alert contains the information of its triggering event.

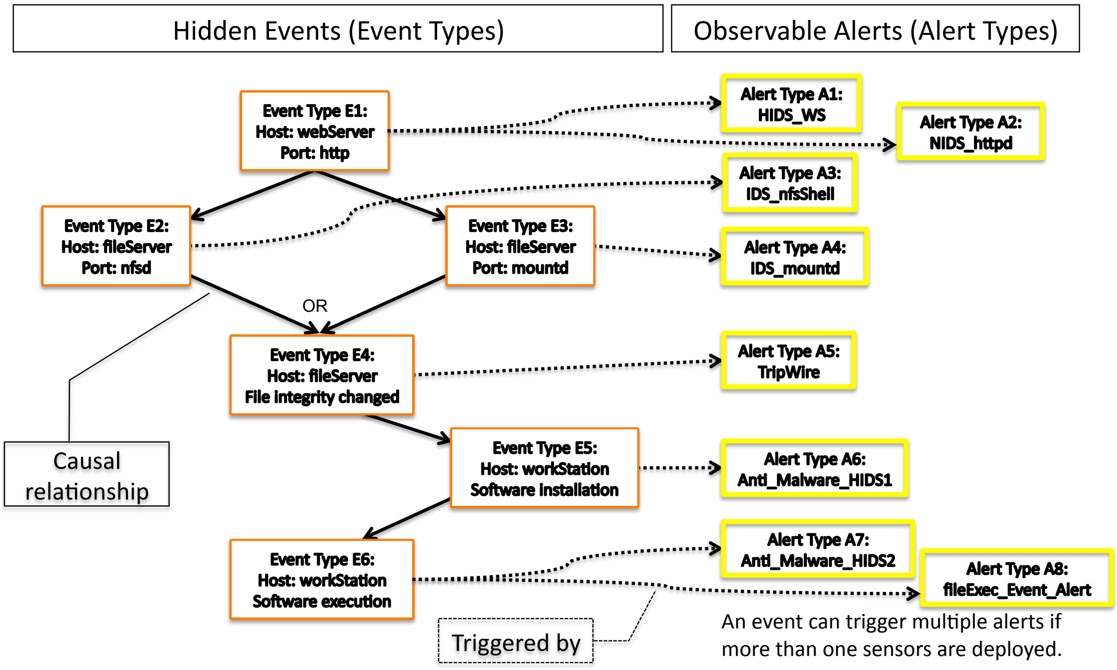

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E2: Host: fileServer Port: nfsd
Event Type E3: Host: fileServer Port: mountd
OR
Event Type E4: Host: fileServer File integrity changed
Event Type E5: Host: workStation Software installation
Event Type E6: Host: workStation Software execution
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A3: IDS_nfsShell
Alert Type A4: IDS_mountd
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A7: Anti_Malware_HIDS2
Alert Type A8: fileExec_Event_Alert
Causal relationship
Triggered by
An event can trigger multiple alerts if more than one sensors are deployed.

Figure 3.6. Events are linked by causal relationships.

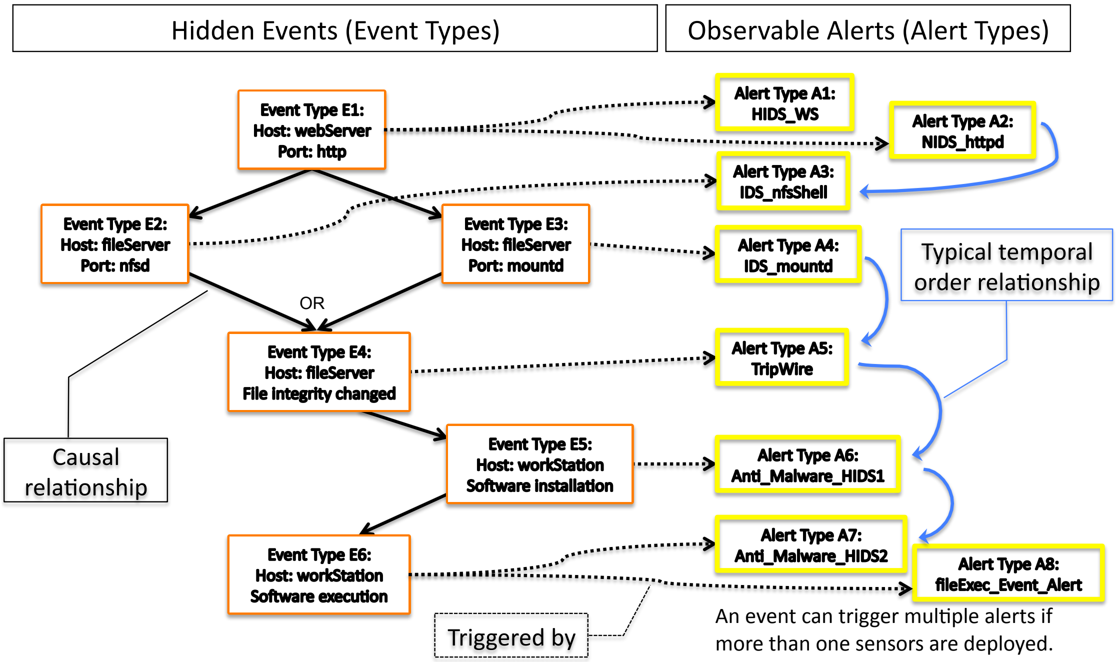

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E2: Host: fileServer Port: nfsd
Event Type E3: Host: fileServer Port: mountd
OR
Event Type E4: Host: fileServer File integrity changed
Event Type E5: Host: workStation Software installation
Event Type E6: Host: workStation Software execution
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A3: IDS_nfsShell
Alert Type A4: IDS_mountd
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A7: Anti_Malware_HIDS2
Alert Type A8: fileExec_Event_Alert
Typical temporal order relationship
Causal relationship
Triggered by
An event can trigger multiple alerts if more than one sensors are deployed.

Figure 3.7. Causal relationships imply typical temporal order of alerts.

indicated by the solid curved lines on the right in Figure 3.7. These components are the important features in our approach to dealing with experience capturing and matching.

3.2.2 Model Specification

The formal specification of the partially observable event-alert system is presented as two parts. The first part is the specification without hidden information, and the second part extends the first part to define a system that contains unobservable information.

3.2.2.1 Specification of the Event-Alert System

Definition An event-alert system S is formalized as a 4-tuple (E, A, C, T), where $E = \{E_1, .., E_m\}$ is a finite set consisting of a number of Event Types, $A = \{A_1, .., A_n\}$ is a finite set consisting of a number of Alert Types, $C = \{C_1, .., C_o\}$ is a set of hyperedges that describe the causality relationship between a set of Event Types and another Event Type, and $T = \{T_1, .., T_p\}$ is a set of links that represents which Event Type can trigger which Alert Type.

Definition An Event Type $E_i \in E$ is a type of event such that a number of event instances can be instantiated based on it. An event instance e_j is an occurrence of the happening of an Event Type.

Definition An Alert Type $A_i \in A$ is a type of alert such that a number of alert instances can be instantiated based on it. An alert instance a_j is an occurrence of an alert being generated.

Definition A causality relationship $C_i = (P, q) \in C$ is a hyperedge with one or more sources and a single target. A causality relationship $C_i = (P, q)$ is composed of one of more preconditions $P = \{p_1, .., p_n\}$ and one postcondition q. The postcondition q can be true only if all the preconditions $P = \{p_1, .., p_n\}$ are all satisfied. $q \rightarrow (p_1 \wedge .. \wedge p_n)$ Each of the conditions refers to an occurrence of a particular Event Type. $\forall d \in P \cap q \; \exists E_d \; ((\exists e \text{ of type } E_d) \Leftrightarrow d)$

Definition A triggering relation $T_i = (E_j, A_k) \in T$ is based on a function $\sigma : E \rightarrow A$ mapping one Event Type to an Alert Type, $E_j \in E$, $A_k \in A$. This function $\sigma : E \rightarrow A$ specifies which Event Type can trigger which Alert Type. An alert can be generated only when an event of its triggering Event Type occurs. ($\exists a$ of type A_k) $\rightarrow$ ($\exists e$ of type E_j))

Example Figure 3.8 is an Event-Alert System $S = (E, A, C, T)$, where
$E = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7\}$,
$A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8\}$,
$C = \{(\{E_1\}, E_3), (\{E_1, E_2\}, E_4), (\{E_3\}, E_5), (\{E_4\}, E_5), (\{E_5\}, E_6), (\{E_6\}, E_7)\}$, and
$T = \{(E_1, A_2), (E_2, A_1), (E_3, A_4), (E_4, A_3), (E_5, A_5), (E_6, A_6), (E_7, A_7), (E_7, A_8)\}$.
Note that instances of E_4 would occur only when instances of both E_1 and E_2 had already occurred. Instance of E_5 would occur only when instances of either E_3 or E_4 had occurred.

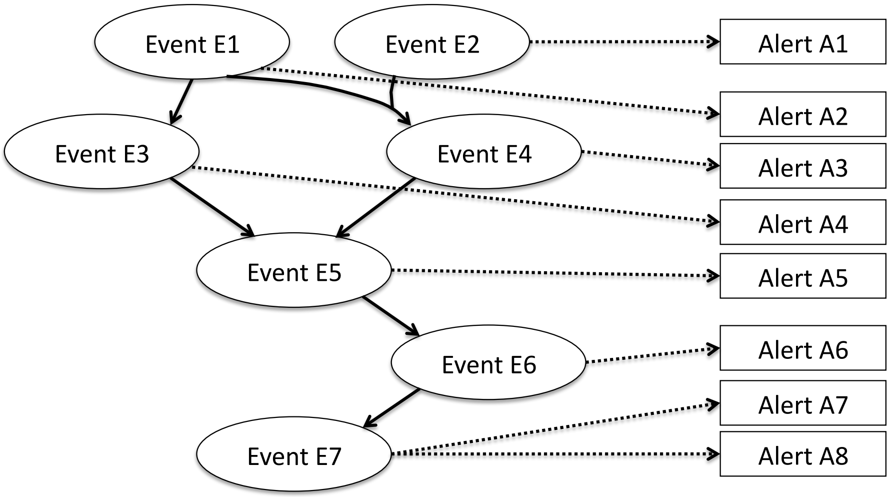


Figure 3.8. An example event-alert system.

In addition to the structure, each event instance or alert instance has a time stamp property that indicates the time of its occurrence. With these time stamps, the time interval between each pair of event instances can be calculated, and the delay between each alert and its triggering event can be determined as well.

3.2.2.2 Partially Observable Event-Alert System

Based on the above definition, the partially observable event-alert system is defined as follows.

Definition A partially observable event-alert system P is an event-alert system where all alert instances are observable to the user, but all types of the events are hidden from the user. Although the events are hidden, they can still be indirectly observed through their corresponding alerts if the alerts are available.

Example An example of a partially observable event-alert system is the attack detection framework in the domain of cyber-security. Given a network configuration including all of the networking devices, firewall rules, software running on the hosts, and other relevant settings, an attack graph can be depicted. An attack graph consists of a number of Event Types with their preconditions and postconditions. The instances of these Event Types are typically not directly observable to the user, requiring that we install sensors such as intrusion detectors or malware scanners to generate observable alerts. Figure 3.9 illustrates a partially observable event-alert system in which all Event Types are hidden.

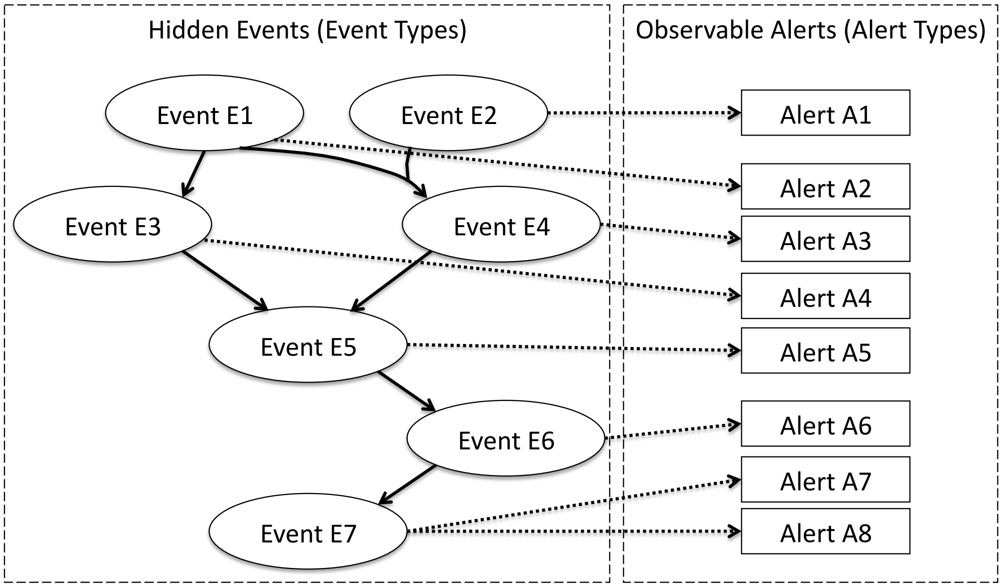


Figure 3.9. An example of the partially observable event-alert system.

3.2.3 Run-Time Information from a Partially Observable Event-Alert System

Given a partially observable event-alert system, the run-time information from the system can be formalized as the following. Since what is observable to the analyst are alerts but not events, the run-time information delivered to the analyst is an alert sequence.

Definition Given a partially observable event-alert system $S = (E, A, C, T)$, there is an alert sequence $q =< a_1, .., a_n >$ being generated at run-time. Each instance of alert $a_i = \{T_A, t_A, T_E, t_E\}$ contains the following information:
T_A: the type of this alert instance,
t_A: the time stamp for when this alert becomes available to the user,
T_E: the type of the event which triggers this alert instance, and
t_E: the time stamp for when the hidden event occurs.

Example Below is an example of a run-time alert sequence from one of the attack approaches on the partially observable event-alert system presented in 3.2.1. Each time stamp is represented as a double point value.
$q =<(A_2, 100.954952318985, E_1, 100.444106202458)$,
$(A_1, 102.008834601172, E_1, 100.444106202458)$,
$(A_4, 102.206178481854, E_3, 101.957002238603)$,
$(A_5, 102.568711110576, E_4, 101.970140070745)$,
$(A_6, 104.333991432356, E_5, 103.954473587865)$,
$(A_8, 104.450035476412, E_6, 104.190414163726)$,
$(A_7, 104.961320179469, E_6, 104.190414163726)>$

3.3 Experience Capturing

Much like the process of training a naïve decision maker to become an expert, an experience base is built through an increasing fashion. It is cumulative, expanding as new incidents are learned. A more experienced expert will have a larger number of past instances in his or her mind to retrieve while facing an on-going situation. The process of experience capturing is illustrated in Figure 3.10. After an intrusion

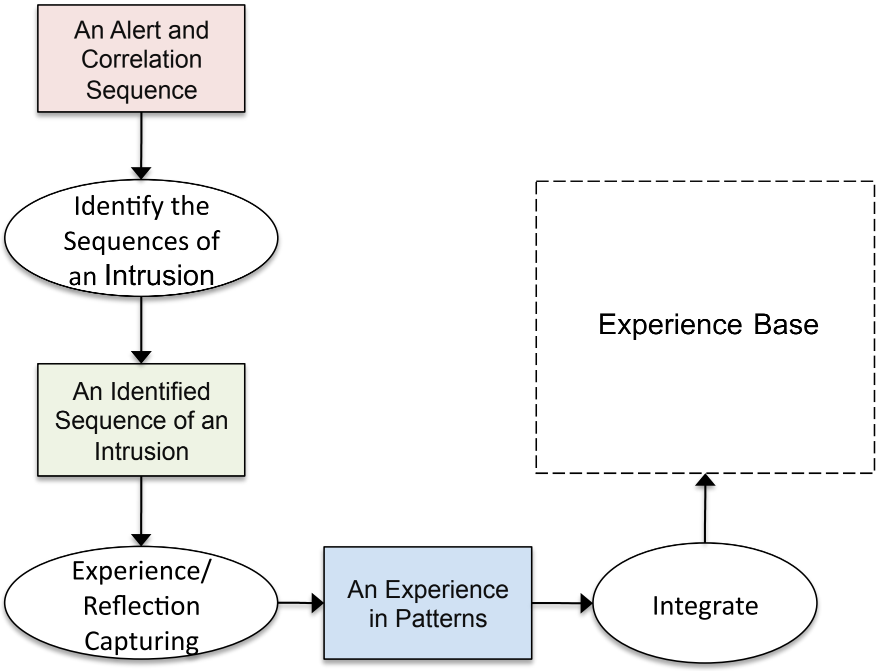

An Alert and Correlation Sequence
Identify the Sequences of an Intrusion
An Identified Sequence of an Intrusion
Experience/ Reflection Capturing
An Experience in Patterns
Integrate
Experience Base

Figure 3.10. The process of experience capturing.

is identified, the run-time sequence will be captured and transformed into a predefined knowledge representation that will form a pattern for future recognition. For experience representation, two specific approaches—Horn rules and regular expressions—will be addressed in this section.

3.3.1 Experience Patterns as Horn Rules

To address the problem of cyber situation recognition based on the partially observable event-alert model, the following items are important and need to be included in the pattern:

- Type declarations of alerts and triggering events
- Temporal relationships of the alerts
- Temporal relationships of the triggering events
- Alert correlation information

The approach we propose is to use forward-chaining rules to represent experience patterns. Forward-chaining rule is based on Horn logic, which is flexible and easy to understand. It not only has powerful expressiveness but also has a friendly structure for knowledge engineers. In addition, existing rule processing algorithms such as Rete [103] and LEAPS [104] have been proven very useful. Consequently, Horn rule is an appropriate candidate for representing experience patterns.

In order to capture the signatures of an experience, we defined two separate patterns for each experience. One is the "Event Pattern," which captures hidden events. The typical temporal order among hidden events is implied by their causal relationships and is of great importance. The other is the "Alert Pattern," which captures observable alerts. Alerts are also important because they are the clues seen by the analyst.

3.3.1.1 Predicates for Representing Experience Patterns

In order to capture event patterns into Horn rules, a list of predicates is defined as follows. These are based on the scenario in 3.2.1, which contains six types of events and eight types of alerts. We also assume that the time stamps of all events in the distributed system are totally ordered according to logical clocks [105], such that we can use a unique time stamp as the identifier of an event.

- A predicate $E1_T$ for declaring the event happened at time T is of the type E_1. Similarly, predicates $E2_T$, .., $E6_T$ indicate events of types E_2, .., E_6.
- Comparison predicates $<_{X,Y}$, $>_{X,Y}$, etc. for describing temporal relationships such as the "happen-before" relation between two events
- A predicate $Recognition_{ExpName,\ PatternType,\ Status}$ for making an assertion about the status of a recognition

Based on these predicate definitions, we can create event patterns as shown in the following example. Note that strings with a precedent question mark (?) are variables.

- $E1_{?e1} \wedge E3_{?e3} \wedge E4_{?e4} \wedge E5_{?e5} \wedge E6_{?e6}$
  $\wedge <_{?e1,?e3} \wedge <_{?e3,?e4} \wedge <_{?e4,?e5} \wedge <_{?e5,?e6}$
  $\Longrightarrow\ Recognition_{experience1,\ eventPattern,\ recognized}$

For alert patterns, additional predicates are defined as follows:

- Predicates $A1_T$, .., $A8_T$ for alerts of types A_1, .., A_8 happened at time T.

With these predicate definitions, an alert pattern can be represented as:

- $A2_{?a2} \wedge A1_{?a1} \wedge A4_{?a4} \wedge A5_{?a5} \wedge A6_{?a6} \wedge A8_{?a8} \wedge A7_{?a7}$
  $\wedge <_{?a2,?a1} \wedge <_{?a1,?a4} \wedge <_{?a4,?a5} \wedge <_{?a5,?a6} \wedge <_{?a6,?a8} \wedge <_{?a8,?a7}$
  $\Longrightarrow \ Recognition_{experience1,\ alertPattern,\ recognized}$

3.3.1.2 Incorporating Alert Correlation Information

In addition to observable alerts themselves, alert correlation information generated by alert correlation engines at run-time also plays an important role. It can indicate which alerts are correlated, but it cannot guarantee which alerts are not correlated. The reports from alert correlation engines are tuples that indicate which alerts are correlated. The number of elements in each tuple typically varies from two to seven; for example, the results of a 2-gram alert correlation engine can be presented as pairs.

Alert correlation information is important because it can be viewed as meta information about alerts and help reduce false recognition due to false positive alerts. For simultaneously occuring attacks, correlation information can provide important clues for distinguishing one attack from another.

Correlation reports are generated at run-times by alert correlation engines. They form part of the run-time situation and should be taken into consideration during the recognition process. Consequently, the correlation information should also be incorporated into experience patterns. It is important to remember that because there is no guarantee that a correlation engine can detect all correlations in the alert sequence, an analyst must consider the possibility that some alerts may triggered by the same attack but not reported by the engine.

In order to incorporate correlation information into the pattern, one more predicate is defined as follows:

- Predicates $Correlation_{G,T}$ for declaring a correlation group G and the alert T is a member of it. For example, the following assertions are sufficient to define the correlation group $(a1, a3, a6)$:

$Correlation_{group1,a1}$
$Correlation_{group1,a3}$
$Correlation_{group1,a6}$

With additional correlation information $(A2_{?a2}, A1_{?a1})$, $(A8_{?a8}, A7_{?a7})$, the previous example alert pattern can be redefined as follows:

- $A2_{?a2} \wedge A1_{?a1} \wedge A4_{?a4} \wedge A5_{?a5} \wedge A6_{?a6} \wedge A8_{?a8} \wedge A7_{?a7}$
  $\wedge <_{?a2,?a1} \wedge <_{?a1,?a4} \wedge <_{?a4,?a5} \wedge <_{?a5,?a6} \wedge <_{?a6,?a8} \wedge <_{?a8,?a7}$
  $\wedge Correlation_{?c0,?a2} \wedge Correlation_{?c0,?a1}$
  $\wedge Correlation_{?c1,?a8} \wedge Correlation_{?c1,?a7}$
  $\Longrightarrow \; Recognition_{experience1,\ alertPattern,\ recognized}$

3.3.1.3 Algorithms for Processing Forward Chaining Rules

One of the advantages of using forward chaining rules is the existence of plentiful research on how to process the rules. The Rete algorithm [103] (and its successors) has been recognized as an efficient way to deal with rules. It is a mechanism that exchanges memory usage for run-time responsiveness. It introduces a network data structure for pattern matching, consisting of α nodes, β nodes, and terminal nodes. A rule can be compiled into a tree-like structure using these nodes, such that at run-time the new information can be fed into it in appropriate α nodes which will then trigger internal tokens passing through the β nodes. A rule will be fired if an internal token reaches the terminal nodes.

We can leverage this approach to develop a system with reasonably good performance. For example, if the Rete algorithm method is selected, a rule will be transformed into a Rete tree. At run-time, when we receive an alert, we send it as a token into each tree. If a rule is fired, we notify the user that there is an incident being recognized. Moreover, when the Rete algorithm is adopted, we can carefully arrange the antecedents of a rule to achieve better performance. An added benefit for experiences with slightly different conditions is that the node sharing strategy can be applied in order to further reduce computational load.

3.3.2 Experience Patterns as Regular Expressions

An alternative way to applying rules is to use regular expression to represent experience patterns. The use of regular expression [106] provides a flexible and efficient way to solicit experiences from domain experts. In addition, existing algorithms for token matching with regular expressions can be adopted [107][108][109][110][111] to support pattern matching.

Regular expression is a neat mechanism with concrete and easy-to-understand standards. Its simplicity makes it easy to write. Regular expression was originally designed for matching strings which consist of a list of tokens, but if we can manipulate the meaning of the tokens, it is possible to leverage this mechanism to encode experience patterns and perform situation recognition.

Given an event or alert sequence from an experience instance, we can extract the type of each event or alert and create a unique token or substring for it in the regular expression. A specific set of alphabets is defined to represent event or alert types, and the temporal relationships are implicitly encoded as their order in the regular expression. For the run-time event or alert sequence, the same token manipulation is applied on each event or alert. As an example, the following regular expressions can serve as the patterns in an experience, capturing the signature of that experience in terms of the types and their happening order. Here we leverage the wild card character "." along with the meta-character "*" for matching zero or more occurrences of precedent tokens to allow other alerts to appear in the sequence.

- Event pattern:
  $.* \ E_1 \ .* \ E_3 \ .* \ E_4 \ .* \ E_5 \ .* \ E_6 \ .*$

- Alert pattern:
  $.* \ A_2 \ .* \ A_1 \ .* \ A_4 \ .* \ A_5 \ .* \ A_6 \ .* \ A_8 \ .* \ A_7 \ .*$

Regular expression is a powerful tool. Patterns based on regular expression are neat and easy to maintain. However, it is doubtful that its expressive power is strong enough to solve problems with interleaved incidents, as addressed in the next section.

3.3.2.1 Limitations of the Regular Expression Approach

Regular expression has the advantage of being able to encode structural strings into a relatively simple sentence; however, some default limitations make it inapplicable to more complicated problem domains. In the realm of cyber security, for instance, multiple attacks may occur in an interleaved manner. In order to distinguish one attack from another, alert correlation is critical and should be incorporated into experience as part of the alert pattern. For a run-time alert sequence, the extra correlation information can be viewed as a property of an alert. Therefore, each instance of alert has three main properties: type, time stamp, and correlation assignment. The correlation assignment can be as simple as a number representing the group. It is practical to use alphabets to represent types, and the order of the alphabets can be used to implicitly encode the temporal order. However, since regular expression is a specification of string patterns consisting of alphabets but is not designed to handle meta-information or properties of individual tokens, it is not expressive enough to capture additional properties beyond these.

On the other hand, although meta-information is not supported in the current regular expression or its extensions, alternative ways exist to deal with meta-information and with the properties of individual tokens of a run-time alert sequence: 1) check the correlation constraints after getting a match; and 2) incorporate it into the matching automaton, a common implementation for matching regular expressions. However, both approaches introduce severe computational overhead.

Using the first approach (checking the correlation constraints after getting a match) requires the matcher—usually a deterministic finite automaton (DFA) or a nondeterministic finite automaton (NFA)—to report which specific tokens (i.e., the sub-sequence) in the sequence contribute to the acceptance of the automaton. Many combinations may exist that satisfy for the objective. For example, if we use a regular expression $r =$“$.*\ A\ .*\ B\ .*\ C\ .*\ D\ .*$” to match against the run-time sequence $s =$“$ACBCCCDCD$”, all of the sub-sequences sufficient to contribute to the acceptance are as follows:

$\underline{A}C\underline{BC}CC\underline{D}CD$

$\underline{A}C\underline{BC}CCDC\underline{D}$

$\underline{A}C\underline{B}C\underline{C}C\underline{D}CD$

$\underline{A}C\underline{B}C\underline{C}CDC\underline{D}$
$\underline{A}C\underline{B}CC\underline{CD}CD$
$\underline{A}C\underline{B}CC\underline{C}DC\underline{D}$
$\underline{A}C\underline{B}CCCD\underline{CD}$

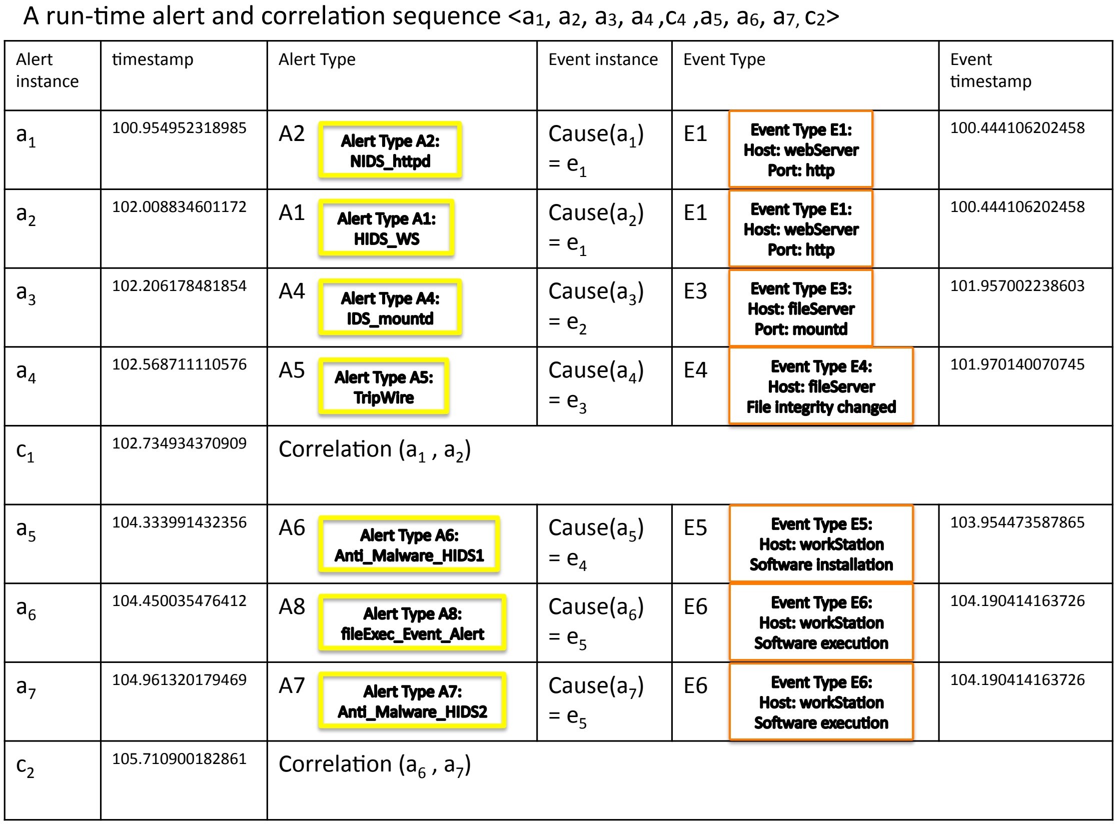
However, existing algorithms will only report a result saying it is matched by running through the first sub-sequence "$\underline{A}C\underline{BC}CC\underline{D}CD$". The purpose of regular expression matching is to decide whether a pattern exists in the sequence but not to report all possible combinations. Therefore, the result does not actually contain useful information for further processing of the correlation information.

Using the second approach (incorporating correlation information into the automaton), the automaton must still record every possible matching path, either by branching or by backtracking, which obviously introduces a lot of run-time overhead. In addition, a correlation report only becomes available *after* all its alerts have occurred; thus it is difficult to decide which outgoing transition to select when the automaton receives an alert.

Regular expression is useful for dealing with pure strings with neither meta-information nor properties in individual tokens. If all of the alerts in a run-time sequence are from the same attack, or if an alert correlation engine can do a perfect job, regular expression should be workable, since it does not need to incorporate meta-information to distinguish interleaved attacks. However, if the complexity of a problem domain is beyond the capability of regular expression, it is necessary to adopt another approach to directly capture all characteristics of the problem. For systems having interleaved incidents such as the cyber security problem, the rule-based approach is more preferable.

3.3.3 An Example Experience Pattern

An example of a run-time alert sequence with correlation information is shown in Figure 3.11. This table shows a sequence of alerts and two correlation reports. To better illustrate the relationships among these alerts, their hidden events, and correlation information, we arranged the information into a graphical presentation based on the partially observable event-alert model. As shown in Figure 3.12, these observable alerts are arranged on the right according to their order of occurrence from top to bottom; correlated alerts are linked with dashed lines. The hidden



A run-time alert and correlation sequence <a_1, a_2, a_3, a_4 ,c_4 ,a_5, a_6, a_7, c_2>

Alert instance	timestamp	Alert Type	Event instance	Event Type	Event timestamp
a_1	100.954952318985	A2 Alert Type A2: NIDS_httpd	Cause(a_1) = e_1	E1 Event Type E1: Host: webServer Port: http	100.444106202458
a_2	102.008834601172	A1 Alert Type A1: HIDS_WS	Cause(a_2) = e_1	E1 Event Type E1: Host: webServer Port: http	100.444106202458
a_3	102.206178481854	A4 Alert Type A4: IDS_mountd	Cause(a_3) = e_2	E3 Event Type E3: Host: fileServer Port: mountd	101.957002238603
a_4	102.568711110576	A5 Alert Type A5: TripWire	Cause(a_4) = e_3	E4 Event Type E4: Host: fileServer File integrity changed	101.970140070745
c_1	102.734934370909	Correlation (a_1 , a_2)			
a_5	104.333991432356	A6 Alert Type A6: Anti_Malware_HIDS1	Cause(a_5) = e_4	E5 Event Type E5: Host: workStation Software installation	103.954473587865
a_6	104.450035476412	A8 Alert Type A8: fileExec_Event_Alert	Cause(a_6) = e_5	E6 Event Type E6: Host: workStation Software execution	104.190414163726
a_7	104.961320179469	A7 Alert Type A7: Anti_Malware_HIDS2	Cause(a_7) = e_5	E6 Event Type E6: Host: workStation Software execution	104.190414163726
c_2	105.710900182861	Correlation (a_6 , a_7)			

Figure 3.11. An example alert and correlation sequence.

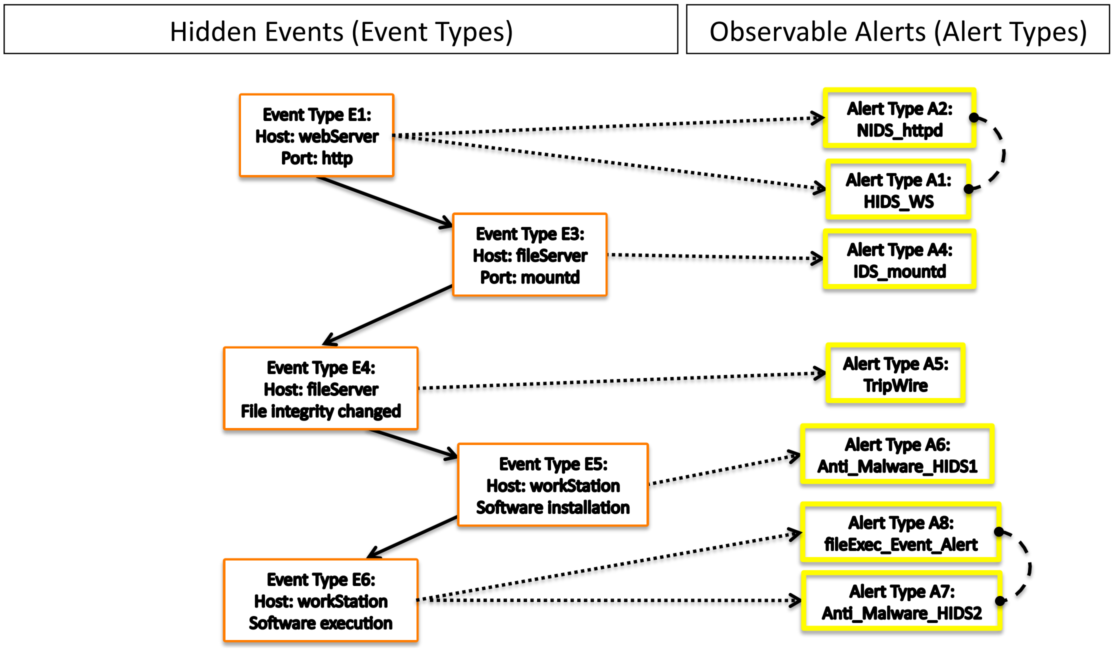
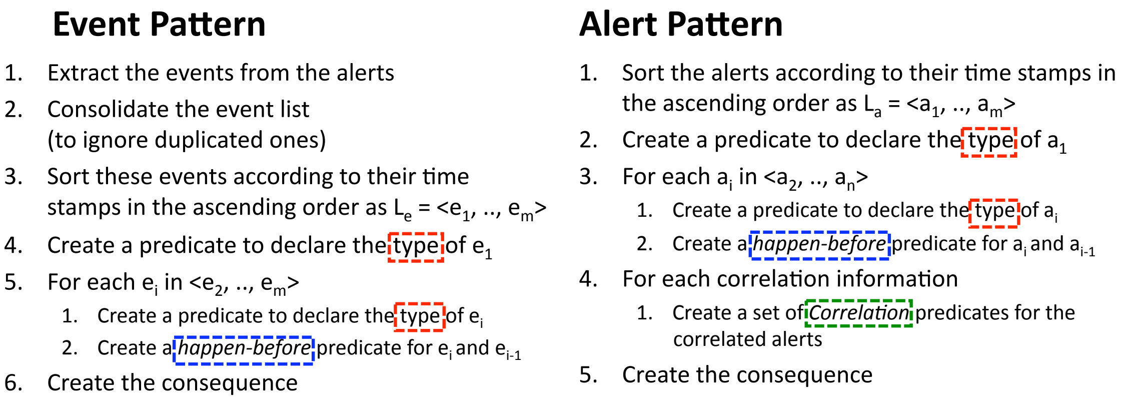

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Alert Type A2: NIDS_httpd
Alert Type A1: HIDS_WS
Event Type E3: Host: fileServer Port: mountd
Alert Type A4: IDS_mountd
Event Type E4: Host: fileServer File integrity changed
Alert Type A5: TripWire
Event Type E5: Host: workStation Software installation
Alert Type A6: Anti_Malware_HIDS1
Alert Type A8: fileExec_Event_Alert
Event Type E6: Host: workStation Software execution
Alert Type A7: Anti_Malware_HIDS2

Figure 3.12. An example alert and correlation sequence presented as a graph.

events are presented on the left with their causal relationships.

To structure alert sequences with correlation information into rule patterns, an algorithm as outlined in Figure 3.13 was developed to automatically construct the patterns given a run-time sequence. Accordingly, the example alert sequence can be converted into both an event pattern and an alert pattern, as shown in Figure 3.14. Variables are denoted as a string with a precedent question mark "?". For better readability, predicates for describing temporal relationships were elaborated using "happen-before" predicates.



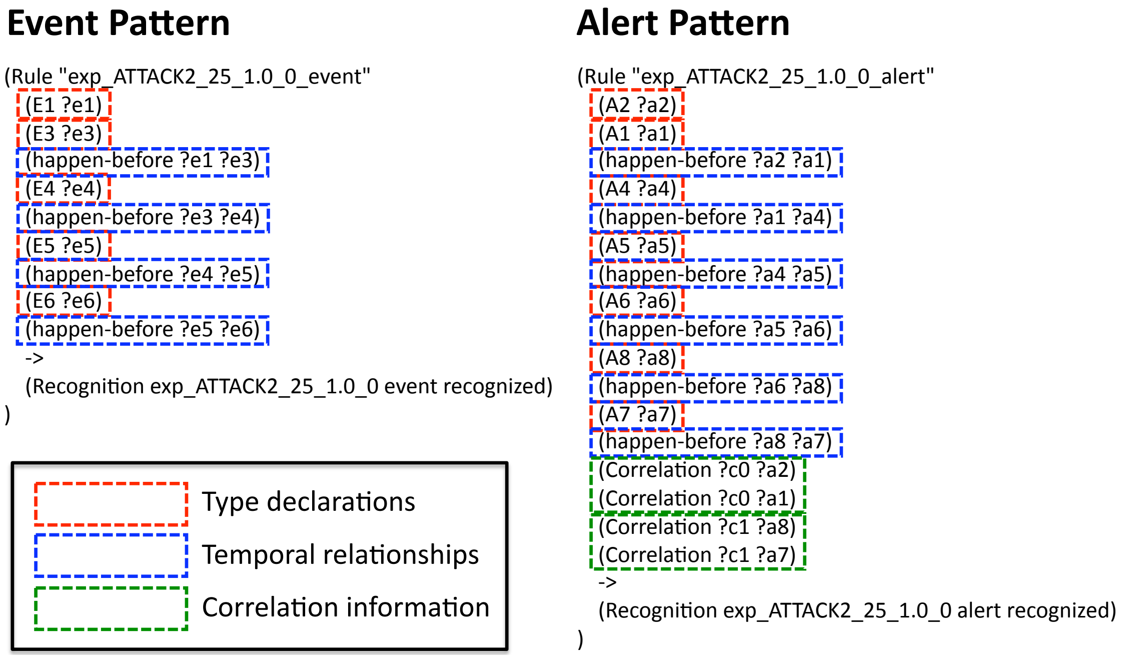
Event Pattern

```
1. Extract the events from the alerts
2. Consolidate the event list
   (to ignore duplicated ones)
3. Sort these events according to their time
   stamps in the ascending order as $L_e = <e_1, .., e_m>$
4. Create a predicate to declare the type of $e_1$
5. For each $e_i$ in $<e_2, .., e_m>$
    1. Create a predicate to declare the type of $e_i$
    2. Create a happen-before predicate for $e_i$ and $e_{i-1}$
6. Create the consequence
```

Alert Pattern

```
1. Sort the alerts according to their time stamps in
   the ascending order as $L_a = <a_1, .., a_m>$
2. Create a predicate to declare the type of $a_1$
3. For each $a_i$ in $<a_2, .., a_n>$
    1. Create a predicate to declare the type of $a_i$
    2. Create a happen-before predicate for $a_i$ and $a_{i-1}$
4. For each correlation information
    1. Create a set of Correlation predicates for the
       correlated alerts
5. Create the consequence
```

Figure 3.13. An algorithm for automatic rule pattern construction.



Event Pattern

```
(Rule "exp_ATTACK2_25_1.0_0_event"
  (E1 ?e1)
  (E3 ?e3)
  (happen-before ?e1 ?e3)
  (E4 ?e4)
  (happen-before ?e3 ?e4)
  (E5 ?e5)
  (happen-before ?e4 ?e5)
  (E6 ?e6)
  (happen-before ?e5 ?e6)
  ->
  (Recognition exp_ATTACK2_25_1.0_0 event recognized)
)
```

Alert Pattern

```
(Rule "exp_ATTACK2_25_1.0_0_alert"
  (A2 ?a2)
  (A1 ?a1)
  (happen-before ?a2 ?a1)
  (A4 ?a4)
  (happen-before ?a1 ?a4)
  (A5 ?a5)
  (happen-before ?a4 ?a5)
  (A6 ?a6)
  (happen-before ?a5 ?a6)
  (A8 ?a8)
  (happen-before ?a6 ?a8)
  (A7 ?a7)
  (happen-before ?a8 ?a7)
  (Correlation ?c0 ?a2)
  (Correlation ?c0 ?a1)
  (Correlation ?c1 ?a8)
  (Correlation ?c1 ?a7)
  ->
  (Recognition exp_ATTACK2_25_1.0_0 alert recognized)
)
```

Figure 3.14. An example experience in Horn rules.

3.4 Situation Recognition

Through experience capturing, more and more experiences are integrated into the experience base. This base can be leveraged to support situation recognition of on-going activities. By comparing a current situation to data within the experience base, similar situations from the past can be retrieved and reported to the analyst.

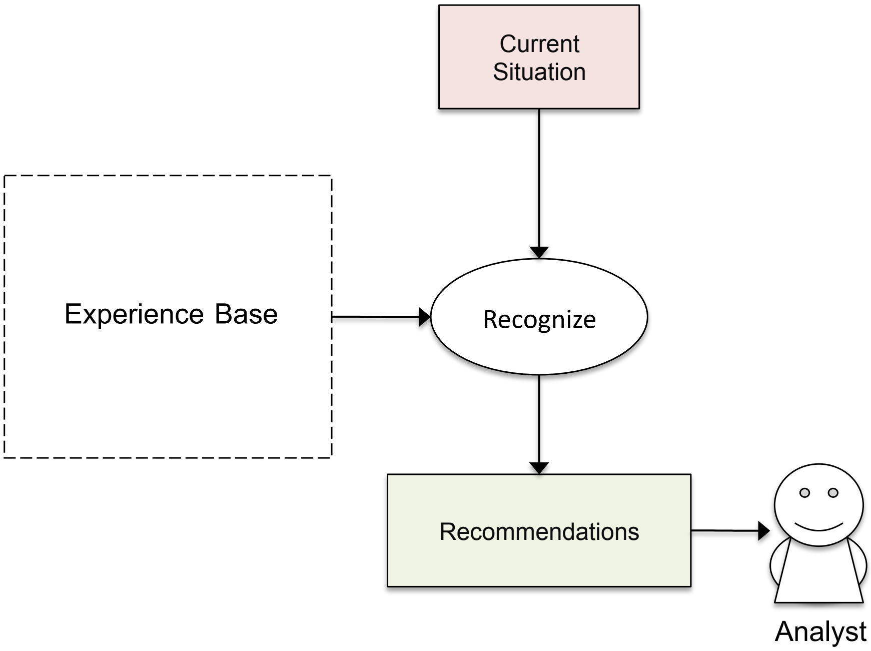


Figure 3.15. The process of situation recognition.

The process of situation recognition is illustrated in Figure 3.15. This is a high-level view of the situation recognition process. The details in design and implementation may vary according to the underlying knowledge representation approach, but the common goal is the same: to make recommendations based on the finding of past experiences which are close to those of the current situation.

3.5 Experience Relaxation

As additional experiences are captured, we may leverage the experience base to identify and analyze future intrusions. However, the size of the experience base may be small, especially in its beginning stage. Therefore, we must address this issue: How can we make a limited number of experiences useful for helping to detect similar situations? To answer this question, we propose the concept of **experience relaxation**.

Strictly speaking, experience is an occurrence of a past incident. It includes every specific detail at a specific moment, such as the exact time and geographical location of the incident; consequently, an experience is unlikely to repeat itself with every single detail remaining the same. Without abstracting these particular details, an experience would become much less useful since its coverage would be restricted to a particular situation. In order to increase the range of applicability, it is desirable to retrieve the important parts of an experience and to relax the experience by trimming those portions which are too specific, such that situation recognition can become more feasible. The higher the degree to which an experience can be relaxed, the higher the possibility exists that it can be matched against a new situation. The concept of experience relaxation is illustrated in Figure 3.16.

With appropriate guidance, experiences generated by relaxation form a hierarchy with the most specific experiences on the top and the most relaxed ones on the bottom. Figure 3.17 shows the concept of level of relaxation. The Level 0 experiences on the top are experience instances with exact information, which are precise but have the narrowest applicability due to their specific descriptions. The Level

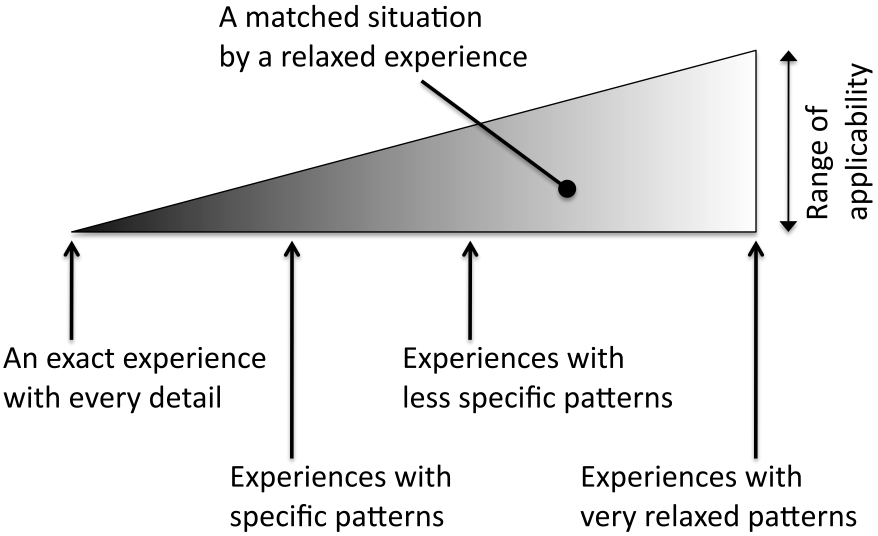


Figure 3.16. Level of experience relaxation.

1 experiences are generated by relaxing less important constraints from the Level 0 experience, which have better applicabilities but are less precise in describing the situation. Experiences at upper levels have better precision, whereas experiences at lower levels provide broader coverage. The entire experience hierarchy is formed through a consistent process, where each level of relaxation is defined with a specification guiding how a higher level experience should be relaxed into lower-level ones. The way in which relaxation is performed should be based on a fundamental model and applied in a systematical way, such that all experience instances on the same level will have a consistent specificity. The model is required to capture important characteristics of the problem domain, such that the relaxation specifications and their priorities can be rationally determined.

The theoretical foundation of experience relaxation is the concept of the relaxable Horn clause [112], which was introduced with the Horn preferential theories in support of constraint logical programming [113][114]. A relaxable Horn clause consists of a head, a body that contains a set of atoms, and a partial ordering among the atoms in the body. This partial ordering represents the priority of the conditions in the body. Conditions with lower priorities may be relaxed if not

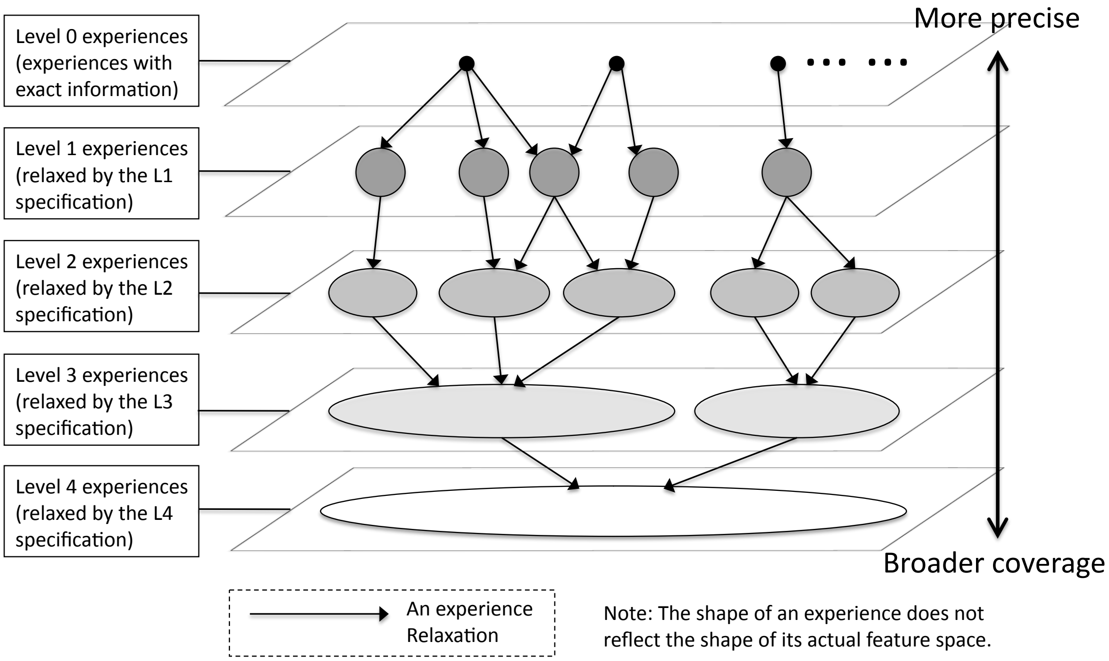

Level 0 experiences (experiences with exact information)
Level 1 experiences (relaxed by the L1 specification)
Level 2 experiences (relaxed by the L2 specification)
Level 3 experiences (relaxed by the L3 specification)
Level 4 experiences (relaxed by the L4 specification)
More precise
Broader coverage
An experience Relaxation
Note: The shape of an experience does not reflect the shape of its actual feature space.

Figure 3.17. A hierarchical experience network.

every condition can be satisfied.

The process of relaxing multiple experience instances allows the construction of a hierarchical experience network, where the experience instances residing in the network coherently form an experience base to facilitate situation recognition. As illustrated in Figure 3.17, one experience may be relaxed into multiple experiences with different coverages, where each one is generated with partial condition relaxation from the original experience. On the other hand, several experiences may be relaxed into identical forms during the relaxation process. This could happen if the relaxation was performed by removing the conditions which serve as the signature of the experience. Figure 3.18 illustrates the process of how a hierarchical experience network is constructed through experience relaxation.

Based on a hierarchical experience network, matching is performed on each experience instance of the network. If multiple experience instances can be retrieved at run-time, the most specific one among all matched instances will be recommended by the system because more constraints will be satisfied, such that the match has a stronger basis. Figure 3.19 demonstrates the flow of situation recognition using a hierarchical experience network.

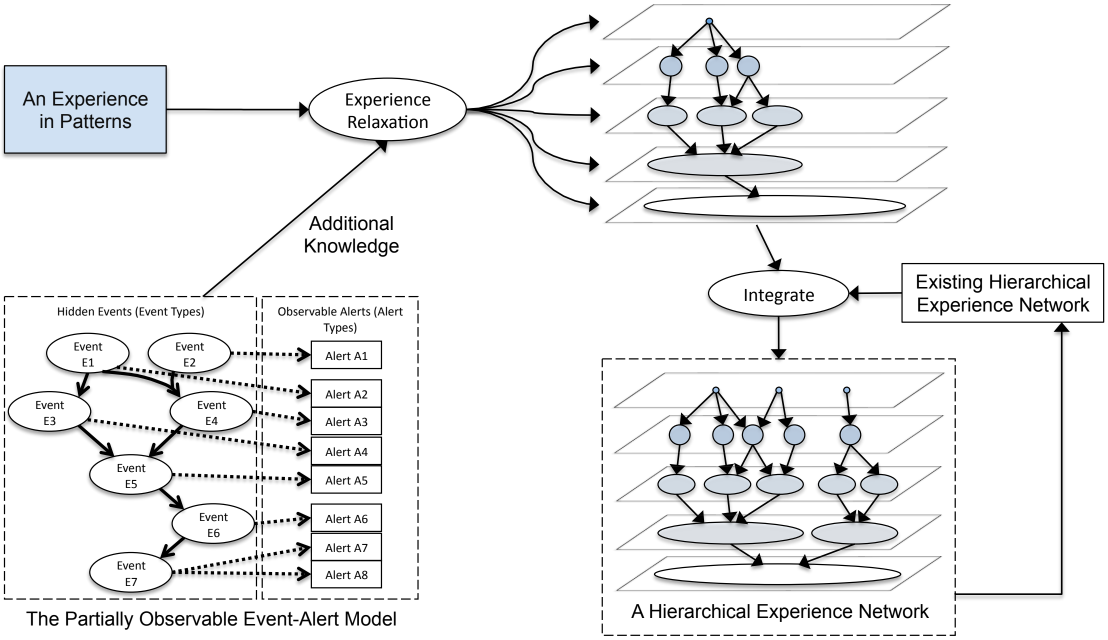

An Experience in Patterns
Experience Relaxation
Additional Knowledge
Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event E1
Event E2
Event E3
Event E4
Event E5
Event E6
Event E7
Alert A1
Alert A2
Alert A3
Alert A4
Alert A5
Alert A6
Alert A7
Alert A8
The Partially Observable Event-Alert Model
Integrate
Existing Hierarchical Experience Network
A Hierarchical Experience Network

Figure 3.18. The construction of a hierarchical experience network.

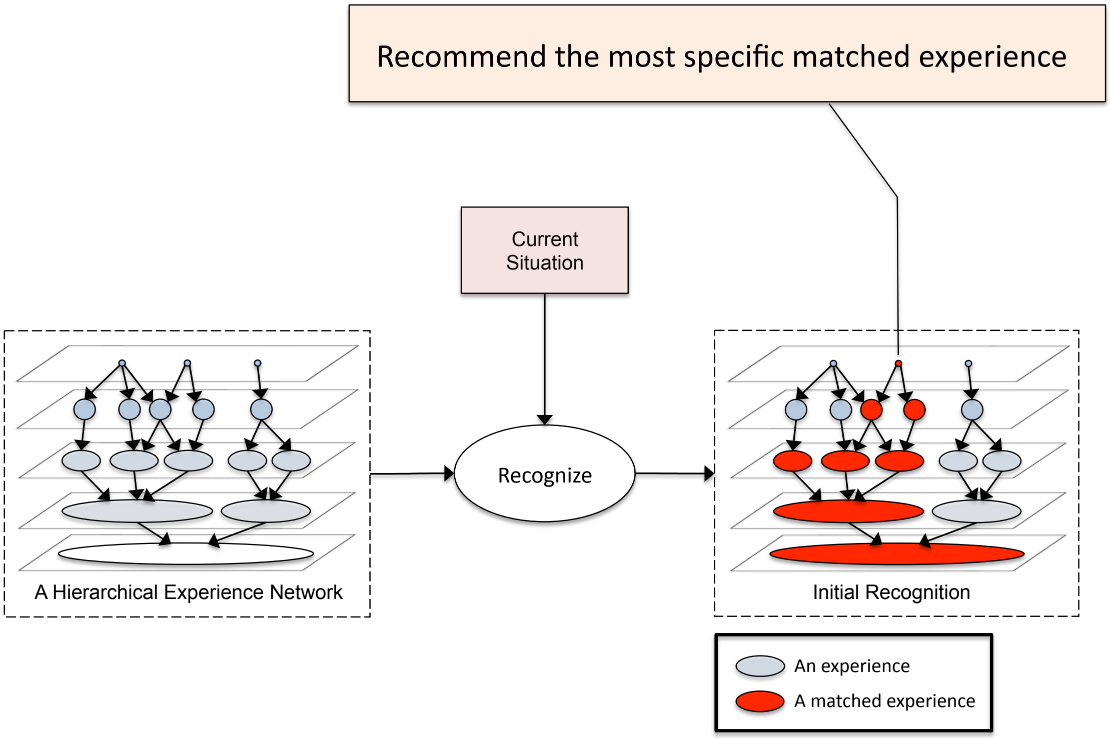

Recommend the most specific matched experience
Current Situation
Recognize
A Hierarchical Experience Network
Initial Recognition
An experience
A matched experience

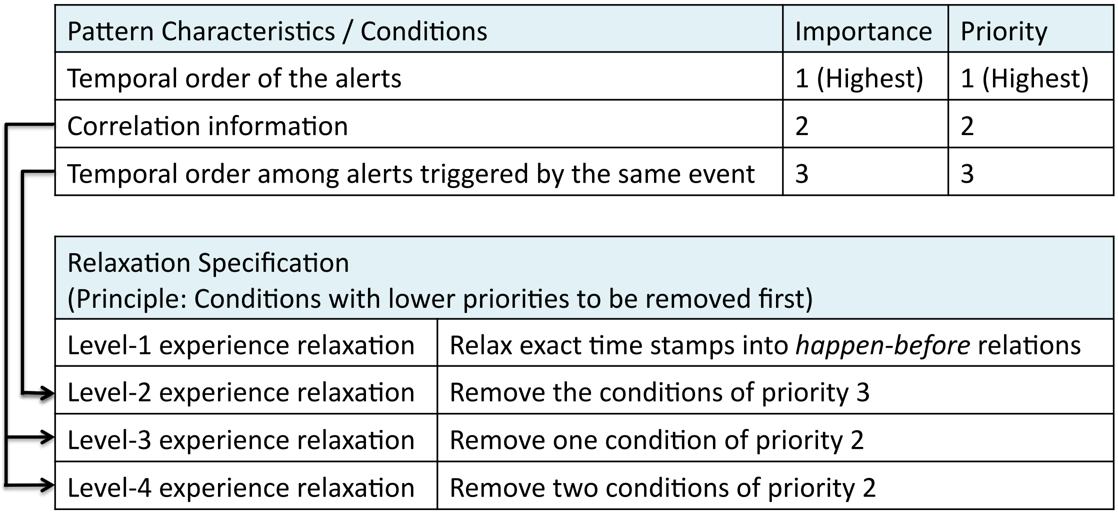
Figure 3.19. Situation recognition using a hierarchical experience network.

3.5.1 Experience Relaxation on Rule Pattern-Based Experiences

In 3.3.1, we showed how experiences can be represented as Horn rules. Here we present how experience relaxation can be applied to experiences in rule patterns.

A rule pattern is composed of a list of conditions in the antecedent. When performing situation recognition using rule pattern-based experiences, each condition in the rule set will be matched against the given situation. If it is not feasible to satisfy all conditions in a rule pattern, we may still obtain a partial matching by relaxing less important conditions.

For rule pattern-based experiences, one way to relax the constraints is to remove one or more conditions from the antecedents of the rule. A relaxed condition set is a logical consequence ($\vdash$) of the condition set it is derived from. The higher the number of conditions that are removed, the wider applicability can be achieved. However, during the process of experience relaxation, a relaxed condition set may lose some degree of the characteristics inherited from its original pattern; in the meantime, we still want to keep the key signature of each experience. Therefore, the importance and priority of each type of condition needs to be carefully determined according to its representativeness and its criticality to the original pattern. Figure 3.20 illustrates an example in which the importance and priority of each characteristic is determined; then, based on those priorities, the relaxation specification can be properly defined.



Pattern Characteristics / Conditions	Importance	Priority
Temporal order of the alerts	1 (Highest)	1 (Highest)
Correlation information	2	2
Temporal order among alerts triggered by the same event	3	3

Relaxation Specification (Principle: Conditions with lower priorities to be removed first)	
Level-1 experience relaxation	Relax exact time stamps into *happen-before* relations
Level-2 experience relaxation	Remove the conditions of priority 3
Level-3 experience relaxation	Remove one condition of priority 2
Level-4 experience relaxation	Remove two conditions of priority 2

Figure 3.20. An example specification for experience relaxation.

According to the relaxation specifications in Figure 3.20, the definition for each level of experience can be organized as follows:

- Level 0 experiences: The original experience instances with specific time stamps.

- Level 1 experiences: Experiences in rules consisting of events and their “happen-before” relationships.

- Level 2 experiences: Based on a Level 1 experience, remove all “happen-before” predicates on the events/alerts whose temporal order does not matter according to their causality relationships. The rationale behind this relaxation is that when temporal order of events is not determined by the causality graph, the order of their alerts is not constrained either. Taking the experience pattern in 3.3.3 as an example, since both alerts $A1$ and $A2$ are triggered by the same event $E1$, the "happen-before" relationship between $A1$ and $A2$ can be removed in order to generate a relaxed experience.

- Level 3 experiences: With one group of correlation constraints removed from a Level 2 experience. Note that there can be multiple Level 3 experiences derived from one Level 2 experience.
- Level 4 experiences: With two groups of correlation constraints removed from a Level 3 experience.
- (Additional levels continue in the same fashion.)

In order to demonstrate the idea of experience relaxation on rule patterns, a series of examples is provided next to show how the experience instance in Section 3.3.3 can be relaxed according to this specification. In addition to the logical form, Figures 3.21 – 3.25 present the same series of experience patterns in the 3.3.3 format, where the predicates for describing temporal relationships are elaborated by using "happen-before" predicates.

- An experience pattern obtained through the Level 1 experience relaxation:
  Event pattern: (Figure 3.21)
  $E1_{?e1} \wedge E3_{?e3} \wedge E4_{?e4} \wedge E5_{?e5} \wedge E6_{?e6}$
  $\wedge <_{?e1,?e3} \wedge <_{?e3,?e4} \wedge <_{?e4,?e5} \wedge <_{?e5,?e6}$
  $\Longrightarrow \; Recognition_{experience1,\ eventPattern,\ recognized}$

  Alert pattern:
  $A2_{?a2} \wedge A1_{?a1} \wedge A4_{?a4} \wedge A5_{?a5} \wedge A6_{?a6} \wedge A8_{?a8} \wedge A7_{?a7}$
  $\wedge <_{?a2,?a1} \wedge <_{?a1,?a4} \wedge <_{?a4,?a5} \wedge <_{?a5,?a6} \wedge <_{?a6,?a8} \wedge <_{?a8,?a7}$
  $\wedge\, Correlation_{?c0,?a2} \wedge Correlation_{?c0,?a1}$
  $\wedge\, Correlation_{?c1,?a8} \wedge Correlation_{?c1,?a7}$
  $\Longrightarrow \; Recognition_{experience1,\ alertPattern,\ recognized}$
- A Level 2 experience pattern obtained by removing the temporal order of alerts from the same event:
  Event pattern: (Figure 3.22)
  $E1_{?e1} \wedge E3_{?e3} \wedge E4_{?e4} \wedge E5_{?e5} \wedge E6_{?e6}$

$\wedge <_{?e1,?e3} \wedge <_{?e3,?e4} \wedge <_{?e4,?e5} \wedge <_{?e5,?e6}$

$\Longrightarrow\ Recognition_{experience1,\ eventPattern,\ recognized}$

Alert pattern:

$A2_{?a2} \wedge A1_{?a1} \wedge A4_{?a4} \wedge A5_{?a5} \wedge A6_{?a6} \wedge A8_{?a8} \wedge A7_{?a7}$

$\cancel{\wedge <_{?a2,?a1}} \wedge <_{?a1,?a4} \wedge <_{?a4,?a5} \wedge <_{?a5,?a6} \wedge <_{?a6,?a8} \cancel{\wedge <_{?a8,?a7}}$

$\wedge\, Correlation_{?c0,?a2} \wedge Correlation_{?c0,?a1}$

$\wedge\, Correlation_{?c1,?a8} \wedge Correlation_{?c1,?a7}$

$\Longrightarrow\ Recognition_{experience1,\ alertPattern,\ recognized}$

- A Level 3 experience pattern obtained by removing one correlation condition:
  Event pattern: (Figure 3.23)

  $E1_{?e1} \wedge E3_{?e3} \wedge E4_{?e4} \wedge E5_{?e5} \wedge E6_{?e6}$

  $\wedge <_{?e1,?e3} \wedge <_{?e3,?e4} \wedge <_{?e4,?e5} \wedge <_{?e5,?e6}$

  $\Longrightarrow\ Recognition_{experience1,\ eventPattern,\ recognized}$

  Alert pattern:

  $A2_{?a2} \wedge A1_{?a1} \wedge A4_{?a4} \wedge A5_{?a5} \wedge A6_{?a6} \wedge A8_{?a8} \wedge A7_{?a7}$

  $\cancel{\wedge <_{?a2,?a1}} \wedge <_{?a1,?a4} \wedge <_{?a4,?a5} \wedge <_{?a5,?a6} \wedge <_{?a6,?a8} \cancel{\wedge <_{?a8,?a7}}$

  $\cancel{\wedge Correlation_{?c0,?a2} \wedge Correlation_{?c0,?a1}}$

  $\wedge\, Correlation_{?c1,?a8} \wedge Correlation_{?c1,?a7}$

  $\Longrightarrow\ Recognition_{experience1,\ alertPattern,\ recognized}$

- A Level 3 experience pattern obtained by removing another correlation condition:
  Event pattern: (Figure 3.24)

  $E1_{?e1} \wedge E3_{?e3} \wedge E4_{?e4} \wedge E5_{?e5} \wedge E6_{?e6}$

  $\wedge <_{?e1,?e3} \wedge <_{?e3,?e4} \wedge <_{?e4,?e5} \wedge <_{?e5,?e6}$

  $\Longrightarrow\ Recognition_{experience1,\ eventPattern,\ recognized}$

  Alert pattern:

  $A2_{?a2} \wedge A1_{?a1} \wedge A4_{?a4} \wedge A5_{?a5} \wedge A6_{?a6} \wedge A8_{?a8} \wedge A7_{?a7}$

  $\cancel{\wedge <_{?a2,?a1}} \wedge <_{?a1,?a4} \wedge <_{?a4,?a5} \wedge <_{?a5,?a6} \wedge <_{?a6,?a8} \cancel{\wedge <_{?a8,?a7}}$

  $\wedge\, Correlation_{?c0,?a2} \wedge Correlation_{?c0,?a1}$

  $\cancel{\wedge Correlation_{?c1,?a8} \wedge Correlation_{?c1,?a7}}$

  $\Longrightarrow\ Recognition_{experience1,\ alertPattern,\ recognized}$

- A Level 4 experience pattern obtained by removing two correlation conditions:

  Event pattern: (Figure 3.25)

  $E1_{?e1} \wedge E3_{?e3} \wedge E4_{?e4} \wedge E5_{?e5} \wedge E6_{?e6}$

  $\wedge <_{?e1,?e3} \wedge <_{?e3,?e4} \wedge <_{?e4,?e5} \wedge <_{?e5,?e6}$

  $\Longrightarrow \ Recognition_{experience1,\ eventPattern,\ recognized}$

  Alert pattern:

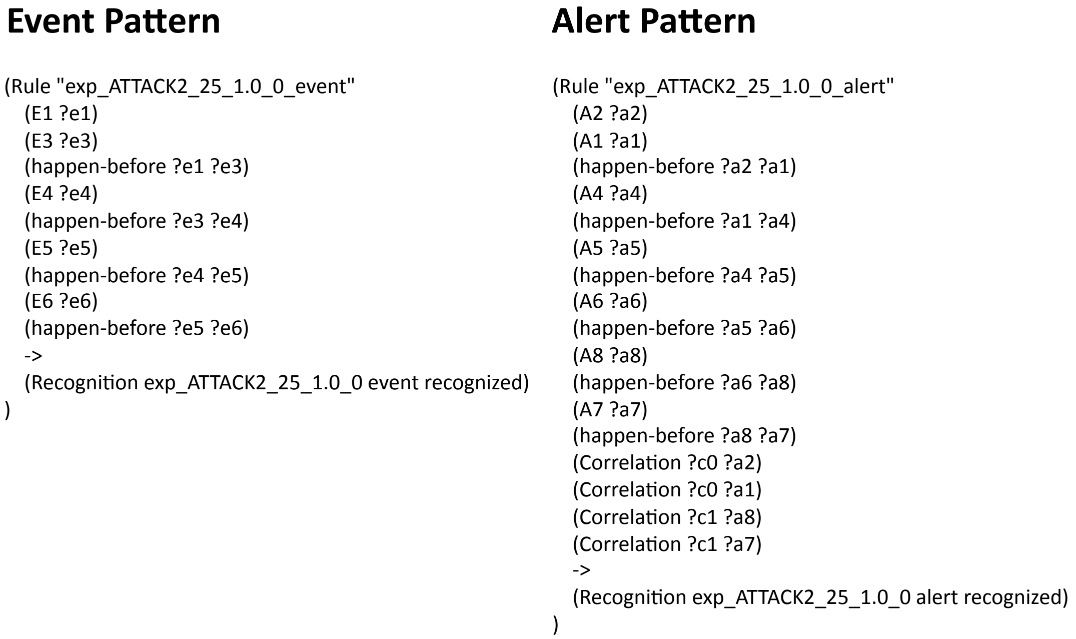
  $A2_{?a2} \wedge A1_{?a1} \wedge A4_{?a4} \wedge A5_{?a5} \wedge A6_{?a6} \wedge A8_{?a8} \wedge A7_{?a7}$

  $\cancel{\wedge <_{?a2,?a1}} \wedge <_{?a1,?a4} \wedge <_{?a4,?a5} \wedge <_{?a5,?a6} \wedge <_{?a6,?a8} \cancel{\wedge <_{?a8,?a7}}$

  $\cancel{\wedge Correlation_{?c0,?a2} \wedge Correlation_{?c0,?a1}}$

  $\cancel{\wedge Correlation_{?c1,?a8} \wedge Correlation_{?c1,?a7}}$

  $\Longrightarrow \ Recognition_{experience1,\ alertPattern,\ recognized}$



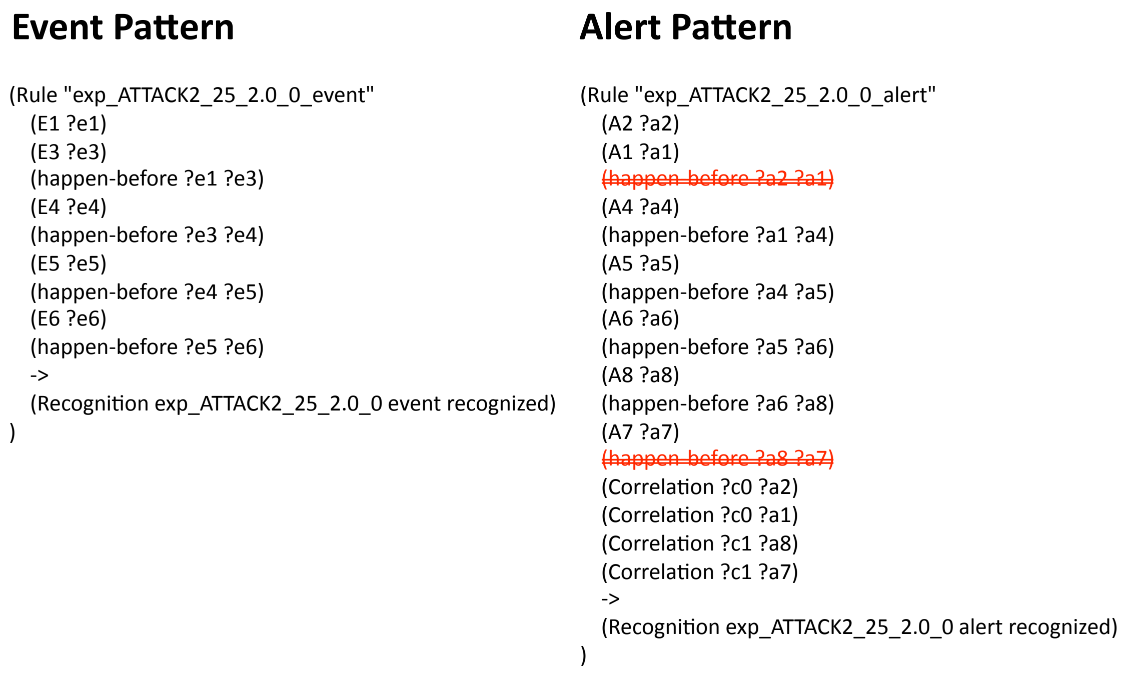
Event Pattern

```
(Rule "exp_ATTACK2_25_1.0_0_event"
  (E1 ?e1)
  (E3 ?e3)
  (happen-before ?e1 ?e3)
  (E4 ?e4)
  (happen-before ?e3 ?e4)
  (E5 ?e5)
  (happen-before ?e4 ?e5)
  (E6 ?e6)
  (happen-before ?e5 ?e6)
  ->
  (Recognition exp_ATTACK2_25_1.0_0 event recognized)
)
```

Alert Pattern

```
(Rule "exp_ATTACK2_25_1.0_0_alert"
  (A2 ?a2)
  (A1 ?a1)
  (happen-before ?a2 ?a1)
  (A4 ?a4)
  (happen-before ?a1 ?a4)
  (A5 ?a5)
  (happen-before ?a4 ?a5)
  (A6 ?a6)
  (happen-before ?a5 ?a6)
  (A8 ?a8)
  (happen-before ?a6 ?a8)
  (A7 ?a7)
  (happen-before ?a8 ?a7)
  (Correlation ?c0 ?a2)
  (Correlation ?c0 ?a1)
  (Correlation ?c1 ?a8)
  (Correlation ?c1 ?a7)
  ->
  (Recognition exp_ATTACK2_25_1.0_0 alert recognized)
)
```

Figure 3.21. An experience pattern obtained through the Level 1 experience relaxation.



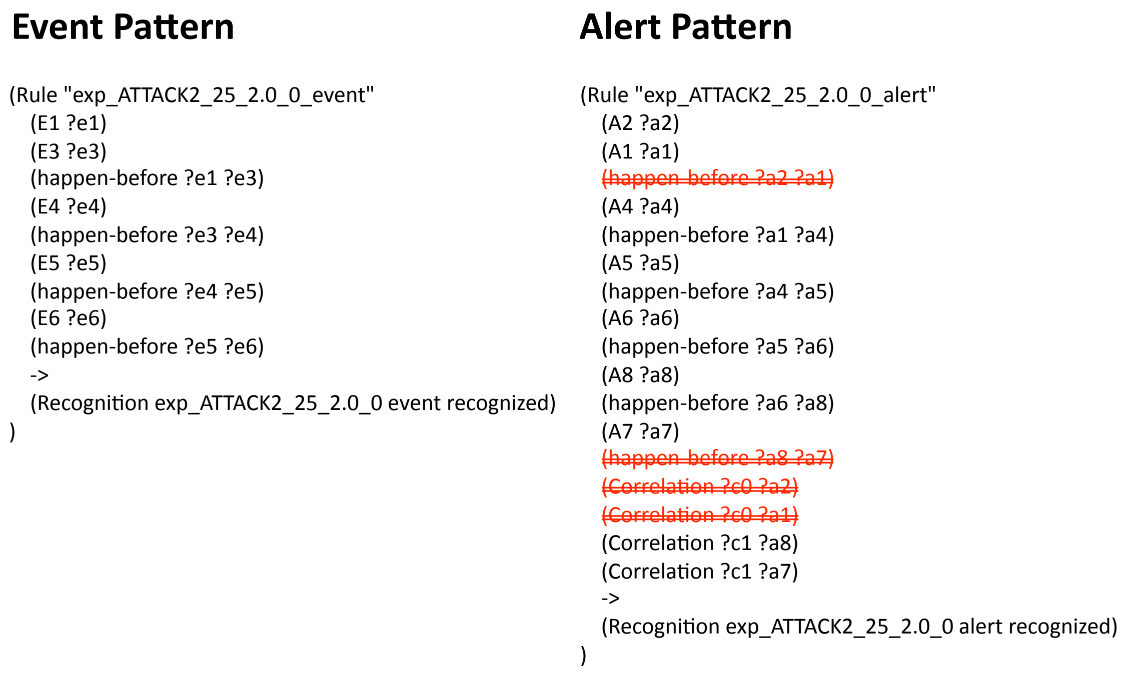
Event Pattern

```
(Rule "exp_ATTACK2_25_2.0_0_event"
  (E1 ?e1)
  (E3 ?e3)
  (happen-before ?e1 ?e3)
  (E4 ?e4)
  (happen-before ?e3 ?e4)
  (E5 ?e5)
  (happen-before ?e4 ?e5)
  (E6 ?e6)
  (happen-before ?e5 ?e6)
  ->
  (Recognition exp_ATTACK2_25_2.0_0 event recognized)
)
```

Alert Pattern

```
(Rule "exp_ATTACK2_25_2.0_0_alert"
  (A2 ?a2)
  (A1 ?a1)
  ~~(happen-before ?a2 ?a1)~~
  (A4 ?a4)
  (happen-before ?a1 ?a4)
  (A5 ?a5)
  (happen-before ?a4 ?a5)
  (A6 ?a6)
  (happen-before ?a5 ?a6)
  (A8 ?a8)
  (happen-before ?a6 ?a8)
  (A7 ?a7)
  ~~(happen-before ?a8 ?a7)~~
  (Correlation ?c0 ?a2)
  (Correlation ?c0 ?a1)
  (Correlation ?c1 ?a8)
  (Correlation ?c1 ?a7)
  ->
  (Recognition exp_ATTACK2_25_2.0_0 alert recognized)
)
```

Figure 3.22. A Level 2 experience pattern obtained by removing the temporal order of alerts triggered by the same event.



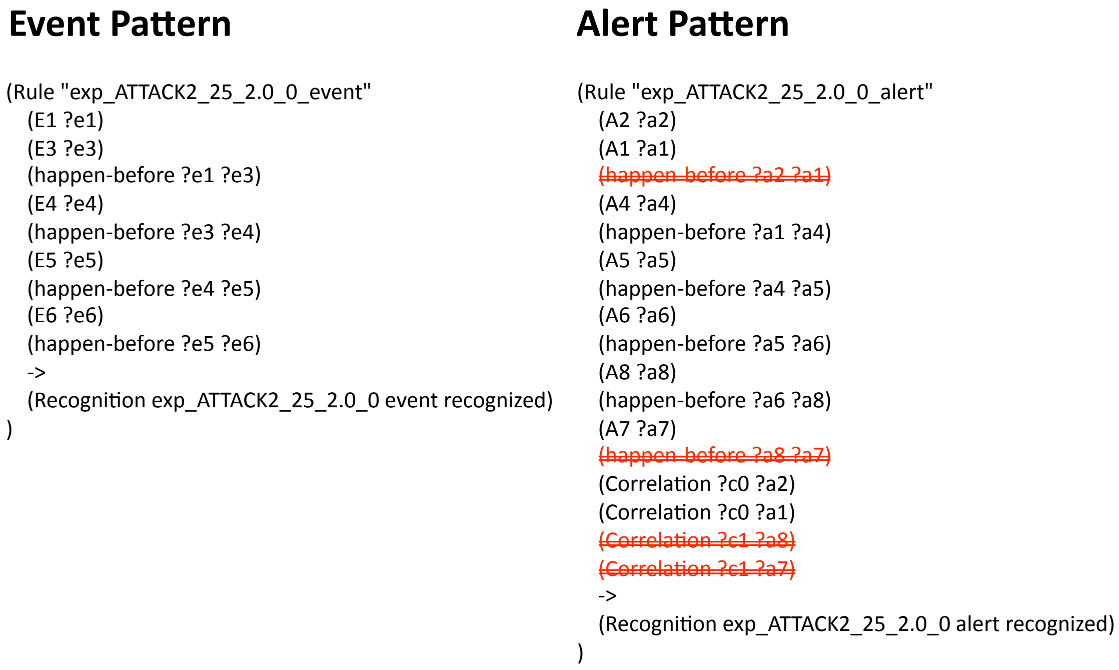
Event Pattern

```
(Rule "exp_ATTACK2_25_2.0_0_event"
  (E1 ?e1)
  (E3 ?e3)
  (happen-before ?e1 ?e3)
  (E4 ?e4)
  (happen-before ?e3 ?e4)
  (E5 ?e5)
  (happen-before ?e4 ?e5)
  (E6 ?e6)
  (happen-before ?e5 ?e6)
  ->
  (Recognition exp_ATTACK2_25_2.0_0 event recognized)
)
```

Alert Pattern

```
(Rule "exp_ATTACK2_25_2.0_0_alert"
  (A2 ?a2)
  (A1 ?a1)
  ~~(happen-before ?a2 ?a1)~~
  (A4 ?a4)
  (happen-before ?a1 ?a4)
  (A5 ?a5)
  (happen-before ?a4 ?a5)
  (A6 ?a6)
  (happen-before ?a5 ?a6)
  (A8 ?a8)
  (happen-before ?a6 ?a8)
  (A7 ?a7)
  ~~(happen-before ?a8 ?a7)~~
  ~~(Correlation ?c0 ?a2)~~
  ~~(Correlation ?c0 ?a1)~~
  (Correlation ?c1 ?a8)
  (Correlation ?c1 ?a7)
  ->
  (Recognition exp_ATTACK2_25_2.0_0 alert recognized)
)
```

Figure 3.23. A Level 3 experience pattern obtained by removing one correlation condition.



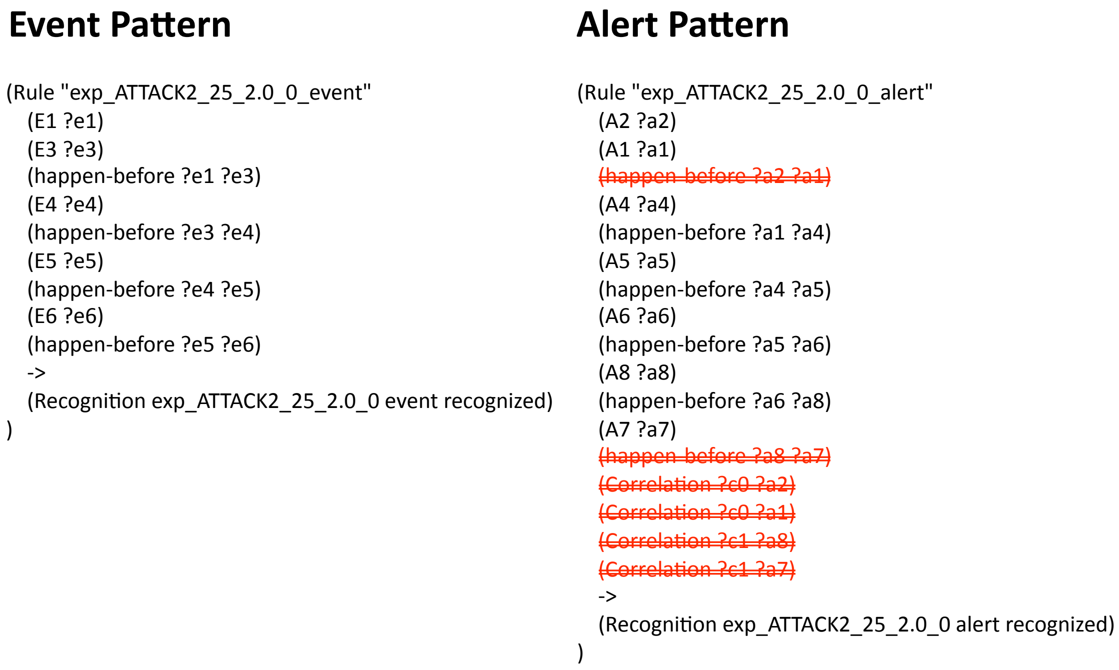
Event Pattern

(Rule "exp_ATTACK2_25_2.0_0_event"
  (E1 ?e1)
  (E3 ?e3)
  (happen-before ?e1 ?e3)
  (E4 ?e4)
  (happen-before ?e3 ?e4)
  (E5 ?e5)
  (happen-before ?e4 ?e5)
  (E6 ?e6)
  (happen-before ?e5 ?e6)
  ->
  (Recognition exp_ATTACK2_25_2.0_0 event recognized)
)

Alert Pattern

(Rule "exp_ATTACK2_25_2.0_0_alert"
  (A2 ?a2)
  (A1 ?a1)
  ~~(happen-before ?a2 ?a1)~~
  (A4 ?a4)
  (happen-before ?a1 ?a4)
  (A5 ?a5)
  (happen-before ?a4 ?a5)
  (A6 ?a6)
  (happen-before ?a5 ?a6)
  (A8 ?a8)
  (happen-before ?a6 ?a8)
  (A7 ?a7)
  ~~(happen-before ?a8 ?a7)~~
  (Correlation ?c0 ?a2)
  (Correlation ?c0 ?a1)
  ~~(Correlation ?c1 ?a8)~~
  ~~(Correlation ?c1 ?a7)~~
  ->
  (Recognition exp_ATTACK2_25_2.0_0 alert recognized)
)

Figure 3.24. A Level 3 experience pattern obtained by removing another correlation condition.



Event Pattern

(Rule "exp_ATTACK2_25_2.0_0_event"
  (E1 ?e1)
  (E3 ?e3)
  (happen-before ?e1 ?e3)
  (E4 ?e4)
  (happen-before ?e3 ?e4)
  (E5 ?e5)
  (happen-before ?e4 ?e5)
  (E6 ?e6)
  (happen-before ?e5 ?e6)
  ->
  (Recognition exp_ATTACK2_25_2.0_0 event recognized)
)

Alert Pattern

(Rule "exp_ATTACK2_25_2.0_0_alert"
  (A2 ?a2)
  (A1 ?a1)
  ~~(happen-before ?a2 ?a1)~~
  (A4 ?a4)
  (happen-before ?a1 ?a4)
  (A5 ?a5)
  (happen-before ?a4 ?a5)
  (A6 ?a6)
  (happen-before ?a5 ?a6)
  (A8 ?a8)
  (happen-before ?a6 ?a8)
  (A7 ?a7)
  ~~(happen-before ?a8 ?a7)~~
  ~~(Correlation ?c0 ?a2)~~
  ~~(Correlation ?c0 ?a1)~~
  ~~(Correlation ?c1 ?a8)~~
  ~~(Correlation ?c1 ?a7)~~
  ->
  (Recognition exp_ATTACK2_25_2.0_0 alert recognized)
)

Figure 3.25. A Level 4 experience pattern obtained by removing two correlation conditions.

3.6 Summary

In this chapter, we discussed the model of partially observable event-alert systems, the process of experience capturing, knowledge representation for experience patterns, the process of situation recognition, and the concept of experience relaxation. These elements are chained together as a complete flow.

As depicted in Figure 3.26, experiences are obtained from identified intrusions and captured as patterns according to a specific knowledge representation. These captured patterns are then processed by experience relaxation and integrated into a hierarchical experience network. This hierarchical experience network can be leveraged during the process of situation recognition. Based on the matching results between the current situation and past experiences, recommendations can be generated and delivered to the analyst. After the analyst makes a decision, the decision made by the analyst can be retained and treated as a new experience for future use, thus forming a complete cycle of the experience-based approach.

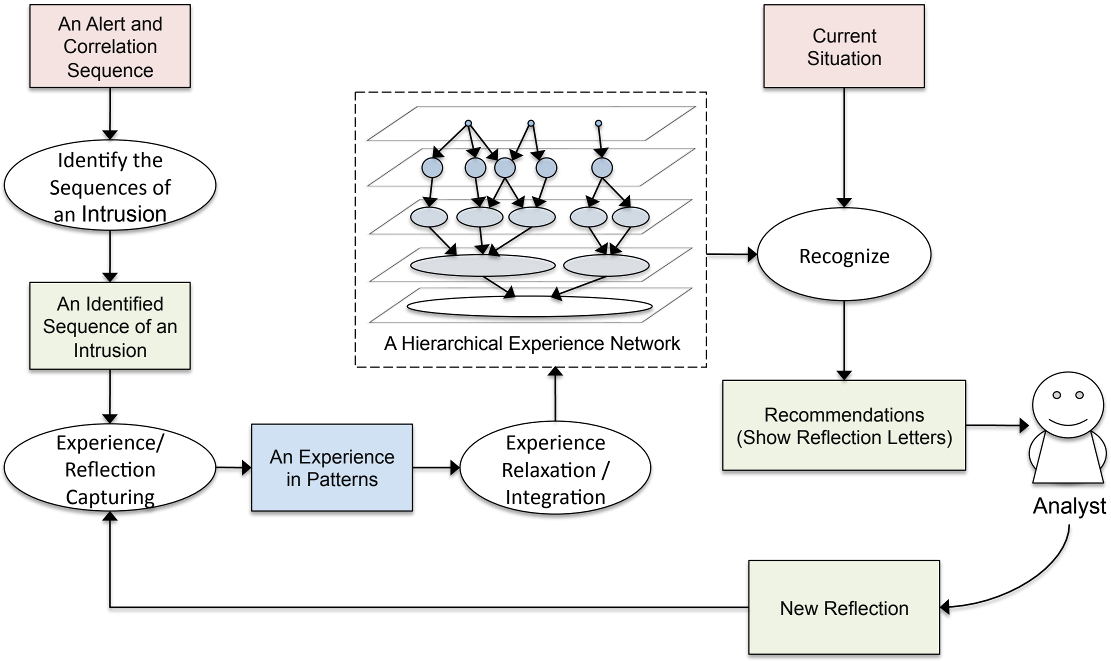

An Alert and Correlation Sequence
Identify the Sequences of an Intrusion
An Identified Sequence of an Intrusion
Experience/ Reflection Capturing
An Experience in Patterns
Experience Relaxation / Integration
A Hierarchical Experience Network
Current Situation
Recognize
Recommendations (Show Reflection Letters)
Analyst
New Reflection

Figure 3.26. The entire process including experience capturing, situation recognition, and experience retainment.

Chapter 4

Implementation of an Agent-Based Cyber Security Decision Support System

In order to realize the decision support mechanism, we implemented a prototype system on top of the R-CAST (RPD-Enabled Collaborative Agents for Simulating Teamwork) multi-agent system [13][12][115]. This chapter reviews the design of this agent-based decision support system and analyzes different parts of the implementation.

4.1 Overview

In our implementation, the experience-based approach was realized on top of R-CAST, a multi-agent system developed using the Java programming language. Its component-based architecture allows a developer to compose an agent according to the specific requirements of a domain problem. Within each agent, different components can communicate with each other through a request whiteboard, which provides a set of interfaces to facilitate intra-agent communications. R-CAST agents can exchange messages with each other via heterogeneous communication protocols such as Web service, Java RMI (Remote Method Invocation), JMS (Java Message Service), and other protocols. Multiple R-CAST agents can team up and share information to perform complicated tasks [116][117]. On top of its architecture, various applications can be developed and overlaid that will solve problems or simulate systems of distributed nature.

In order to realize the experience-based approach, we developed an agent-based system called ER-CAST (Experience-based R-CAST). Figure 4.1 shows the key components of an ER-CAST agent. The large rectangle indicates the boundary of the agent. The Cyber Security Adapter is the interface between external environment and the other internal components. It is responsible for receiving real-time alert and correlation sequences from external information sources. After receiving a piece of information, the Cyber Security Adapter interprets it and transforms it into an internal data structure. This reorganized information will be delivered to the Recognizer. Then the Recognizer will perform cyber situation recognition by consulting the Knowledge Base, which is composed of an experience base and a rule processing system. Finally, after an intrusion is identified, recommendations will be sent to the users.

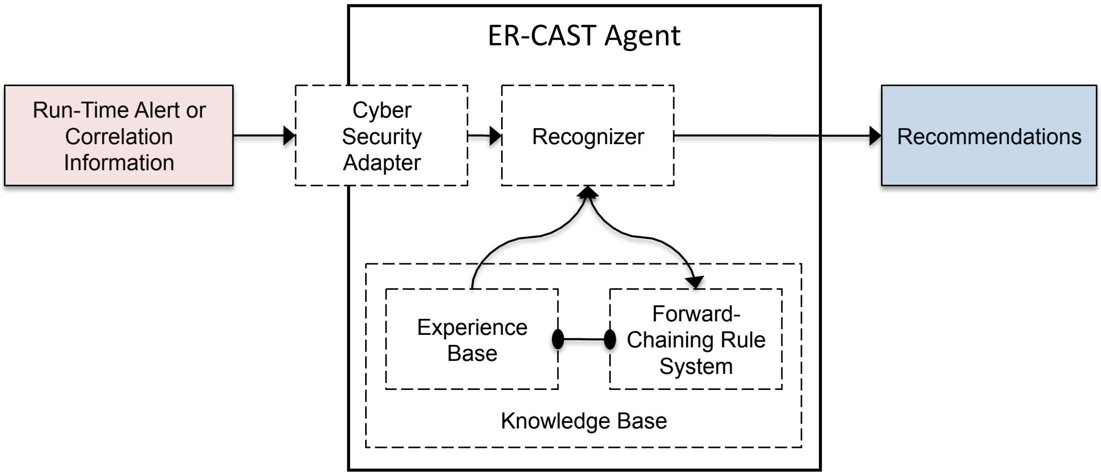


Figure 4.1. An ER-CAST agent.

In the following sections, we cover more details about experience representation, the Recognizer, and the rule processing system. In addition, we also introduce two additional decision modules that can be added to provide more advanced functionalities. The supporting services that facilitate agent management will be presented at the end of the chapter.

4.2 Representation of Experiences

According to the discussion in Chapter 3, an experience can be described as a combination of an event pattern, an alert pattern, a class label representing the type of intrusion, and meta information regarding the level of relaxation of this experience. The definition of an experience is given as the following:

Definition An experience based on our model is defined as $e = (EP, AP, L, G)$, where EP is the event pattern for describing the occurrence of the hidden events, AP is the alert pattern for those observable alerts we received, L is the class label of this intrusion, and G is the level of relaxation of this particular experience.

The event pattern should capture not only what types of events occurred in this experience but also the order of their occurrence. In addition, the causal relationships among the event types defined in the partially observable event-alert system should also be considered while we relax the experience. Similarly, the alert pattern should follow the constraints imposed by the causal relationships among their triggering events.

The class label of an intrusion is used to denote what was actually happening behind the signature captured in the event pattern and the alert pattern. It is also the answer we expect to get from the decision support system if the given situation is matched with this experience. At run-time, when feeding the decision support system with a sequence of alerts, we expect the system to recognize the current situation according to the experience base and then to advise the user of this particular information.

The property of level of relaxation indicates what degree the experience was relaxed according to the relaxation specification. It is critical to define a reasonable specification for experience relaxation, which includes identifying appropriate importance for each feature and specifying its corresponding priorities.

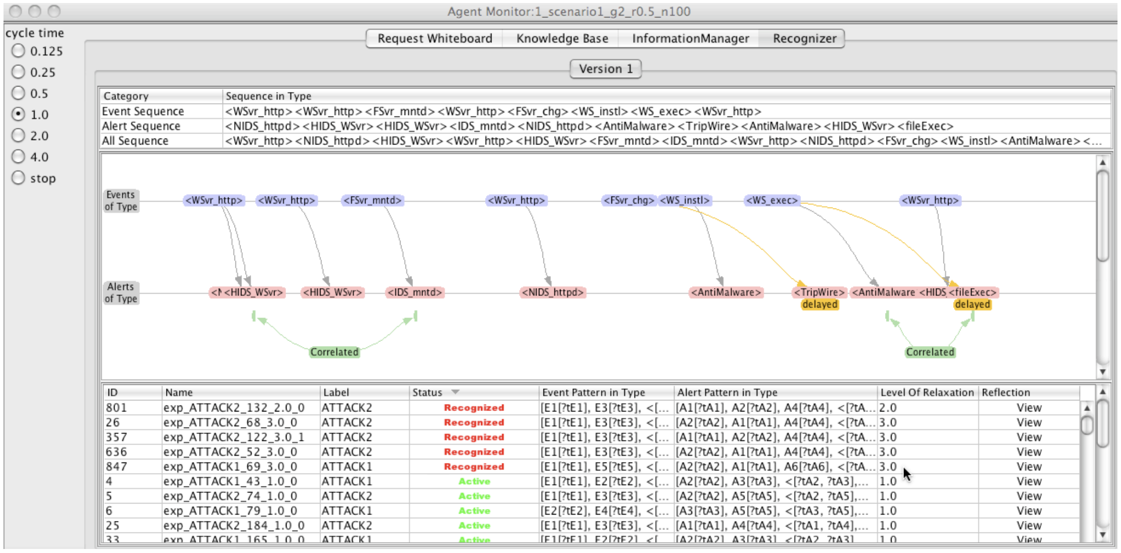
In our implementation, we used Horn rules to represent experiences. For experience relaxation, the specification we employed is the one presented in Figure 3.20.

4.3 The Recognizer

The Recognizer is the component which coordinates the work of experience-based situation recognition. It not only controls the recognition process but also visualizes the current situation for the users.

In the initialization stage, the Recognizer reads the experience base to collect all previous experiences; each experience contains one reference to its event pattern in the rule system and another reference to its alert pattern. During the run time, the incoming alerts and correlation information are sent to the forward-chaining rule system. In the meantime, the Recognizer keeps checking the status of each referred rule in the rule system. For each experience, if both its event rule and its alert rule are fired, the Recognizer will set a flag to indicate that the experience is recognized by the system.

Figure 4.2 is a screenshot of the Recognizer component. There are three panels on its graphical user interface. The top panel lists the alerts and events according to their occurrence, sorted and displayed in terms of their types. The middle panel is a visualization of the event sequence, the alert sequence, and the correlation information, allocated on a two-dimensional plane according to their temporal order. The table on the bottom shows all experience instances in the experience base. Each row is an outline of a specific experience instance, which includes an



Category	Sequence in Type
Event Sequence	<WSvr_http><WSvr_http><FSvr_mntd><WSvr_http><FSvr_chg><WS_instl><WS_exec><WSvr_http>
Alert Sequence	<NIDS_httpd><HIDS_WSvr><HIDS_WSvr><IDS_mntd><NIDS_httpd><AntiMalware><TripWire><AntiMalware><HIDS_WSvr><fileExec>
All Sequence	<WSvr_http><NIDS_httpd><HIDS_WSvr><WSvr_http><HIDS_WSvr><FSvr_mntd><IDS_mntd><WSvr_http><NIDS_httpd><FSvr_chg><WS_instl><AntiMalware><...

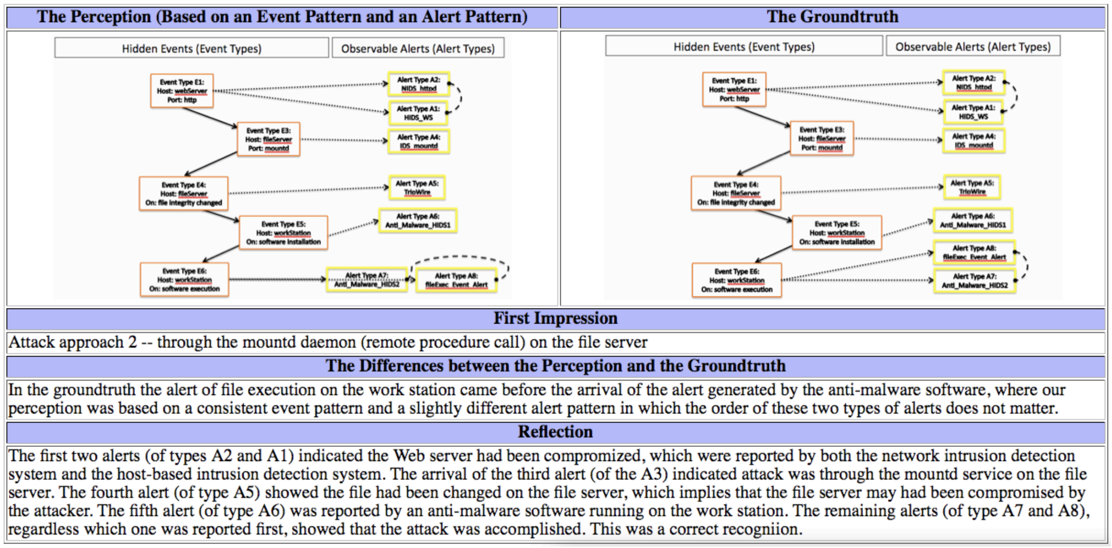
ID	Name	Label	Status	Event Pattern in Type	Alert Pattern in Type	Level Of Relaxation	Reflection
801	exp_ATTACK2_132_2.0_0	ATTACK2	Recognized	[E1[?tE1], E3[?tE3], <[...	[A1[?tA1], A2[?tA2], A4[?tA4], <[?tA...	2.0	View
26	exp_ATTACK2_68_3.0_0	ATTACK2	Recognized	[E1[?tE1], E3[?tE3], <[...	[A2[?tA2], A1[?tA1], A4[?tA4], <[?tA...	3.0	View
357	exp_ATTACK2_122_3.0_1	ATTACK2	Recognized	[E1[?tE1], E3[?tE3], <[...	[A1[?tA1], A2[?tA2], A4[?tA4], <[?tA...	3.0	View
636	exp_ATTACK2_52_3.0_0	ATTACK2	Recognized	[E1[?tE1], E3[?tE3], <[...	[A2[?tA2], A1[?tA1], A4[?tA4], <[?tA...	3.0	View
847	exp_ATTACK1_69_3.0_0	ATTACK1	Recognized	[E1[?tE1], E5[?tE5], <[...	[A2[?tA2], A1[?tA1], A6[?tA6], <[?tA...	3.0	View
4	exp_ATTACK1_43_1.0_0	ATTACK1	Active	[E1[?tE1], E2[?tE2], <[...	[A2[?tA2], A3[?tA3], <[?tA2, ?tA3],...	1.0	View
5	exp_ATTACK2_74_1.0_0	ATTACK2	Active	[E1[?tE1], E3[?tE3], <[...	[A2[?tA2], A5[?tA5], <[?tA2, ?tA5],...	1.0	View
6	exp_ATTACK1_79_1.0_0	ATTACK1	Active	[E2[?tE2], E4[?tE4], <[...	[A3[?tA3], A5[?tA5], <[?tA3, ?tA5],...	1.0	View
25	exp_ATTACK2_184_1.0_0	ATTACK2	Active	[E1[?tE1], E3[?tE3], <[...	[A1[?tA1], A4[?tA4], <[?tA1, ?tA4],...	1.0	View
33	exp_ATTACK1_165_1.0_0	ATTACK1	Active	[E1[?tE1], E2[?tE2], <[...	[A2[?tA2], A3[?tA3], <[?tA2, ?tA3]...	1.0	View

Figure 4.2. A screenshot of the Recognizer component.

identifier, a name, the label showing the type of attack, the recognition status, the event pattern and the alert pattern, the level of relaxation, and a hyperlink to relevant reflection letters. The "Active" status indicates that the recognition procedure is in progress, whereas the status "Recognized" means that the current situation can be recognized based on this particular experience. For each level of relaxation, a lower number means that level is closer to the original experience and has more constraints. The rows in the table are sorted by their recognition status and relaxation level, such that the user can easily identify which experiences are recognized and how close they are to the current situation.

Through the hyperlinks on the far right column, the user can open relevant reflection letters in a Web browser to get more information about past experience. A reflection letter is a documented experience that shows how a past situation was handled. Each reflection letter contains the experience pattern and the groundtruth at that moment, along with the first impression and reflection notes. If reflecting an experience of false recognition, the letter may include notes regarding why recognition failed and how to avoid reiterating the mistake. Figure 4.3 is an example of a reflection letter indicating correct recognition, and Figure 4.4 is an example based on a false recognition.

In sum, the Recognizer controls the recognition process and provides integrated



The Perception (Based on an Event Pattern and an Alert Pattern)

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E3: Host: fileServer Port: mountd
Event Type E4: Host: fileServer On: file integrity changed
Event Type E5: Host: workStation On: software installation
Event Type E6: Host: workStation On: software execution
Alert Type A2: NIDS_httpd
Alert Type A1: HIDS_WS
Alert Type A4: IDS_mountd
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A7: Anti_Malware_HIDS2
Alert Type A8: fileExec_Event_Alert

The Groundtruth

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E3: Host: fileServer Port: mountd
Event Type E4: Host: fileServer On: file integrity changed
Event Type E5: Host: workStation On: software installation
Event Type E6: Host: workStation On: software execution
Alert Type A2: NIDS_httpd
Alert Type A1: HIDS_WS
Alert Type A4: IDS_mountd
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A8: fileExec_Event_Alert
Alert Type A7: Anti_Malware_HIDS2

First Impression

Attack approach 2 -- through the mountd daemon (remote procedure call) on the file server

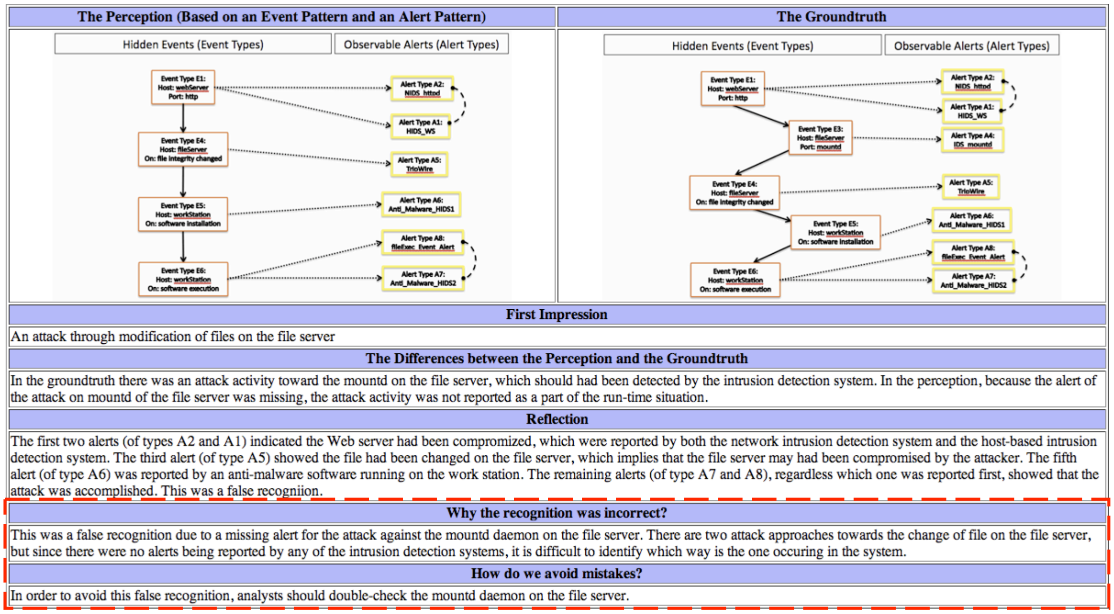
The Differences between the Perception and the Groundtruth

In the groundtruth the alert of file execution on the work station came before the arrival of the alert generated by the anti-malware software, where our perception was based on a consistent event pattern and a slightly different alert pattern in which the order of these two types of alerts does not matter.

Reflection

The first two alerts (of types A2 and A1) indicated the Web server had been compromized, which were reported by both the network intrusion detection system and the host-based intrusion detection system. The arrival of the third alert (of the A3) indicated attack was through the mountd service on the file server. The fourth alert (of type A5) showed the file had been changed on the file server, which implies that the file server may had been compromised by the attacker. The fifth alert (of type A6) was reported by an anti-malware software running on the work station. The remaining alerts (of type A7 and A8), regardless which one was reported first, showed that the attack was accomplished. This was a correct recogniion.

Figure 4.3. An example reflection letter of a correct recognition.



Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E4: Host: fileServer On: file integrity changed
Event Type E5: Host: workStation On: software installation
Event Type E6: Host: workStation On: software execution
Alert Type A2: NIDS_httpd
Alert Type A1: HIDS_WS
Alert Type A5: TrioWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A8: fileExec_Event_Alert
Alert Type A7: Anti_Malware_HIDS2
Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E3: Host: fileServer Port: mountd
Event Type E4: Host: fileServer On: file integrity changed
Event Type E5: Host: workStation On: software installation
Event Type E6: Host: workStation On: software execution
Alert Type A2: NIDS_httpd
Alert Type A1: HIDS_WS
Alert Type A4: IDS_mountd
Alert Type A5: TrioWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A8: fileExec_Event_Alert
Alert Type A7: Anti_Malware_HIDS2

The Perception (Based on an Event Pattern and an Alert Pattern)	The Groundtruth

First Impression

An attack through modification of files on the file server

The Differences between the Perception and the Groundtruth

In the groundtruth there was an attack activity toward the mountd on the file server, which should had been detected by the intrusion detection system. In the perception, because the alert of the attack on mountd of the file server was missing, the attack activity was not reported as a part of the run-time situation.

Reflection

The first two alerts (of types A2 and A1) indicated the Web server had been compromized, which were reported by both the network intrusion detection system and the host-based intrusion detection system. The third alert (of type A5) showed the file had been changed on the file server, which implies that the file server may had been compromised by the attacker. The fifth alert (of type A6) was reported by an anti-malware software running on the work station. The remaining alerts (of type A7 and A8), regardless which one was reported first, showed that the attack was accomplished. This was a false recogniion.

Why the recognition was incorrect?

This was a false recognition due to a missing alert for the attack against the mountd daemon on the file server. There are two attack approaches towards the change of file on the file server, but since there were no alerts being reported by any of the intrusion detection systems, it is difficult to identify which way is the one occuring in the system.

How do we avoid mistakes?

In order to avoid this false recognition, analysts should double-check the mountd daemon on the file server.

Figure 4.4. An example reflection letter of a false recognition.

information through an intuitive interface, allowing it to assist cyber security analysts to make better decisions.

4.4 Implementation of a Rule Processing System

In our implementation, rule experiences are represented and processed in the knowledge base, which is capable of processing forward-chaining rules. In addition, the knowledge base can be regarded as the "brain" of an agent not only for storing and retrieving knowledge but also for providing inference capabilities. The intuitive way of firing forward-chaining rules can be very costly, and algorithms have been developed to address the performance issue; the Rete algorithm [103] significantly improves response time by maintaining a specific tree data structure in memory, whereas LEAPS [104] is based on the concept of relational database systems.

In order to derive a full control of the inference behavior, rather than using existing rule-based systems such as Jess [118], Pellet[119], or Drools [120], a knowledge base was developed to fit into the R-CAST agent architecture. In this section, we start with the basic design of this knowledge base system, followed by descriptions of its other features.

4.4.1 Knowledge Base Design

The R-CAST knowledge base is a forward-chaining rule-based system, which consists of $\mathit{FactTypes}$, Facts, and Rules. FactType is a declaration of the type of predicate. Given a FactType definition, a number of instances called Facts can be instantiated based on it. From the perspective of a relational database, a FactType is similar to a table definition with its properties as the column names and each Fact of the FactType as a row.

Definition A FactType $P_{a_1,..,a_n}$ is an n-ary predicate definition having a_1, .., a_n as its arguments.

Definition A Fact is an instantiation of a FactType $P_{a_1,..,a_n}$ with a particular assignment $\Sigma = \{a_1 = v_1,\ ..,\ a_n = v_n\}$, where each of v_1, .., v_n is a constant value.

Below is an example FactType for the predicate $\mathit{Person}_{?name,?age,?gender}$, which is defined with three arguments—name, age, and gender—where each argument is denoted by a prefix question mark "?" Two example Facts instantiated according to the "Person" FactType definition are listed as well. Their corresponding assignments are $\Sigma_a = \{?name =$ Alice, $?age =$ 30, $?gender =$ female$\}$ and $\Sigma_b = \{?name =$ Bob, $?age =$ 25, $?gender =$ male$\}$ respectively.

```
(FactType Person (?name ?age ?gender))
(Fact     Person (Alice 30   female ))
(Fact     Person (Bob   25   male   ))
```

A Rule is based on Horn logic and is composed of a number of antecedents and a consequent. If all of the antecedents are satisfied under some variable assignment, the rule will be fired to instantiate the consequence as an $\mathit{ImpliedFact}$ using the values from this variable assignment. The definitions of a rule and ImpliedFacts are given as follows:

Definition A Rule $R = \{P, Q, X\}$ is composed of one or more ordered antecedents $P = \{P_1, .., P_n\}$, a consequence Q, and a set of shared variable $X = \{x_1,\ ..,\ x_o\}$ among P_1, .., P_n, and Q. Each of P_1, .., P_n, and Q is a predicate with a particular assignment that contains only constant values or shared variables from X. All variables that appear in the consequence should be bounded by being used in the antecedents. $\forall x_i \in X$, (x_i is used in Q) $\Rightarrow$ (x_i is used in P). A rule is said to be "fired" where there is an assignment Σ' on X that satisfies $P_1 \cap .. \cap P_n$, and then a Fact of Q will be instantiated based on this assignment Σ'.

Definition An $ImpliedFact$ is a Fact that is instantiated due to firing a rule.

The example shown in the following is a forward-chaining rule $R = \{P, Q, X\}$, where $P = \{Person_{?var-name,?var-age,male}, <_{?var-age,12}\}$, $Q = Boy_{?var-name}$, $X = \{?var\text{-}name, ?var\text{-}age\}$.

If there is a $Person$ whose gender is male and his age is less than 12, the rule will be fired to create an instance (Fact) of Boy using the name from the bounded variable $?var\text{-}name$.

```
(Rule (Person (?var-name ?var-age male))
      (< (?var-age 12))
      ->
      (Boy (?var-name)))
```

The syntactic specification of the knowledge base in Extended Backus-Naur Form (EBNF) is outlined in Figure 4.5.

```
KBCONTENT ::= ( FACTTYPE | FACT | RULE )+
---------------------------------------------------------------------------
FACTTYPE  ::= “(FactType ” TYPENAME ” (” PROPERTYNAME+ “) )”
FACT      ::= “(Fact “     TYPENAME ” (” VALUE+ “) )”
RULE      ::= “(Rule “     ANTECEDENTS ”->” CONSEQUENCE “)”
---------------------------------------------------------------------------
TYPENAME       ::= ( character | digit )+
PROPERTYNAME   ::= “?”( character | digit )+
VALUE          ::= ( character | digit )+
ANTECEDENT     ::= CONDITION+
CONDITION      ::= “(“ TYPENAME ( PROPERTYNAME | VALUE )+ “)” |
                   “(“ FUN_PRED ( PROPERTYNAME | VALUE )+ “)”
CONSEQUENCE    ::= “(“ TYPENAME ( PROPERTYNAME | VALUE )+ “)”
FUN_PRED       ::= ”+” | ”-” | ”*” | ”/” | ”mod” | ”pow” |
                   ”=” | ”eq” | ”<” | ”<=” | ”>” | ”>=”
```

Figure 4.5. Syntactic specification of the knowledge base in EBNF.

4.4.2 The Underlying Inference Algorithm

In order to ensure the performance of the forward-chaining rule-based system, the Rete algorithm [103] was implemented to support rule firing. We defined classes of Alpha node, Join node, and Terminal node for the fundamental tree structure as illustrated in Figure 4.6.

Each Alpha node has an associated predicate, which can be viewed as an instantiation of a FactType with unbounded variables. When a fact is asserted, the Alpha node of the same FactType will be the retrieved, and then its associated predicate will be unified with the asserted fact, thus resulting in a set of variable bindings called Contexts. Each Context will then be wrapped as a token to be passed to the outgoing node. A Join node has two queues to accept incoming tokens for matching and then passing merged Context as a token to its outgoing node. Once a Terminal node receives a token, it will create a new ImpliedFact using the bindings retrieved from this token. During retraction, the Rete trees will be traversed in a similar way, but tokens having the same bindings as the retraction sentence will be removed.

Implementing the Rete algorithm ensures the performance of rule firing, but it also introduces some constraints. For example, all assertions/retractions should be initiated only through the designated functions that trigger Rete graph updates, and all the operations should be manipulated in the order consistent with their

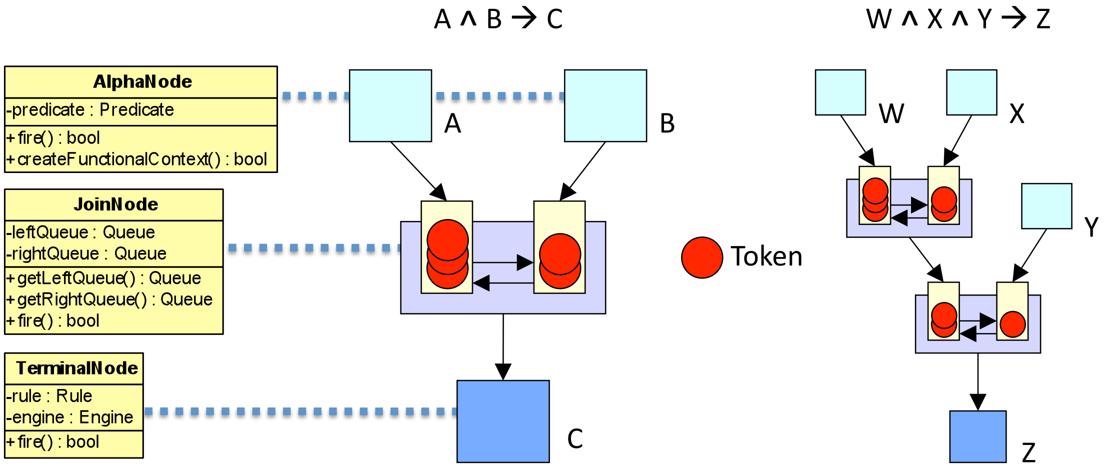


Figure 4.6. An implementation of Rete graphs.

occurring sequence, such that the Rete graph can correctly reflect the current situation. This constraint introduces a drawback: we cannot simply and freely add or delete facts as we wish. Additionally, the design of other features should be handled carefully in order not to break the operations of the Rete algorithm. Every modification to the working memory should be consistent with the Rete graph such that the correctness can be guaranteed.

4.4.3 Truth Maintenance

Firing a rule will create a new ImpliedFact, which in turn may trigger additional rules to create another ImpliedFact and so on. It is critical to manage these dependencies carefully such that while an antecedent becomes invalid, all its dependents will be retracted as well. The cascading retraction is necessary to maintain the validity of the knowledge base.

In order to enable the mechanism of truth maintenance, cross-references are implemented in ImpliedFacts and their supporting facts. As illustrated in Figure 4.7, references of the ImpliedFact are added to the antecedents' dependent lists, and both of the antecedents are together recorded as a *SupporterSet* of the ImpliedFact. The formal definitions of *SupporterSet* and *DependentSet* are as follows.

Definition In a knowledge base, the set of all facts $F = F_{constant} \cap F_{implied}$, where $F_{constant}$ contains facts that are naturally asserted, and $F_{implied}$ contains facts created due to firing a rule. $\forall f_i \in F_{implied}$, $\exists S_i = \{S_{i,1}, .., S_{i,n} | n \geq 1\}$ that contains one or more sets of supporting facts (*SupporterSet*) of f_i, where each *SupporterSet* contains a set of facts which collectively trigger the firing of a rule. $\forall f_i \in F_{implied}$, $\forall S_{i,j} \in S_i$, $S_{i,j} = \{f_k | (f_k \in F) \wedge (f_k \in \text{antecedents of a rule that created } f_i)\}$.

Definition In a knowledge base, the set of all facts $F = F_{constant} \cap F_{implied}$. $\forall f_k \in$

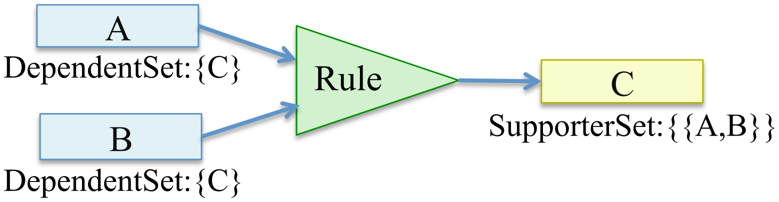


Figure 4.7. Cross-references for supporters-dependent relationships.

F, $\exists D_k = \{f_i | (f_i \in F_{implied}) \wedge (f_k \in \text{antecedents of a rule that created } f_i)\}$ as the *DependentSet* of f_k.

The truth maintenance mechanism is realized by the recursive function in Algorithm 1. After invoking the retraction function in the Rete engine, we check each of its dependent facts and remove the corresponding SupporterSet. The ImpliedFact will need to be retracted as well if it no longer has any SupporterSet.

Algorithm 1 Fact retraction for truth maintenance.

```
Require: A set of Facts $F = F_{constant} \cap F_{implied}$
    $\forall f_i \in F_{implied}$, $\exists S_i = \{S_{i,1}, .., S_{i,n} | n \geq 1\}$
    $\forall f_i \in F_{implied}$, $\forall S_{i,j} \in S_i$, $S_{i,j} = \{f_k | (f_k \in F) \wedge (f_k \in \text{antecedents of a rule that created } f_i)\}$.
    $\forall f_k \in F$, $\exists D_k = \{f_i | (f_i \in F_{implied}) \wedge (f_k \in \text{antecedents of a rule that created } f_i)\}$.
function $RetractFact$ ($f_k$: $Fact$)
$ReteEngine.Retract(f_k)$
Get the DependentSet $D_k$ of $f_k$
for all $f_i \in D_k$ do
    Get the SupporterSet $S_i$ of $f_i$
    for all $S_{i,j} \in S_i$ do
        if $f_k \in S_{i,j}$ then
            Remove $S_{i,j}$ from $S_i$
        end if
    end for
    if $S_i = \{\}$ then
        $RetractFact$ ($f_i$)
    end if
end for
end function
```

4.4.4 Key and Instance Singleton Maintenance

The concept of “key” has been developed and widely used in relational database systems but has not as often appeared in the context of rule-based systems. For some FactTypes in the knowledge base, maintaining Facts with non-duplicated keys is fundamental to the correctness of a system. For example, assuming a person is identified by his/her first name and last name, and each person can be at only one location, the Location FactType can be defined as follows.

```
(FactType Location (?firstName, ?lastName, ?latitude, ?longitude)
              (key (?firstName, ?lastName))
)
```

We declare the combination of ?firstName and ?lastName as the key of the Location FactType, such that for each distinct pair of (?firstName, ?lastName) there can exist only one instance of Location. When the person moves, the old Location should be retracted before the new one is asserted. We call this type of instance an “update-style” predicate.

The implementation is shown in the pseudo-code below. Whenever asserting a fact, we first check whether there is already a fact with the same key. If such a fact exists, it should be removed before the new assertion is made.

```
Function: void handleUpdateStyleFact(Fact F):
   Identify the FactType of F as T;
   Retrieve the key values K from F according to T’s definition;
   Make a query for T with key arguments filled with K;
   For each query result R {
      If (R is different from F) {
         Invoke retractFact(R);
      }
   }
   ReteEngine.Assert(F);
```

4.4.5 Processing Streaming Data

Before introducing update-style predicates into the system, assertions will only trigger assertions, and retractions will only trigger retractions; these can easily handled by immediately applying the appropriate following assertions or retractions. However, with update-style predicates, an assertion may trigger a retraction, and this retraction may lead to other cascading retractions; moreover, the assertion itself may also initiate other cascading assertions, which can also cause further retractions. The situation becomes much more complicated due to these mixed assertions and retractions.

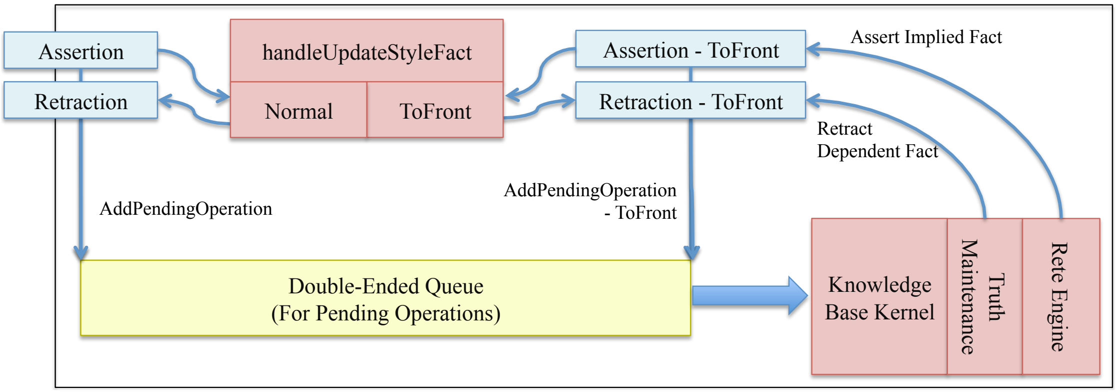


Figure 4.8. The double-ended queue for assertion/retraction management.

To address this issue, a double-ended queue (illustrated in Figure 4.8) was designed to manage complicated assertion-retraction combinations. For a new assertion or retraction, the external function at the top left is invoked, and the operation is added to the end of the queue. If cascading operations are needed, the handleUpdateStyleFact function appends them to the end of the queue in the appropriate order. For those pending operations already in the queue, the knowledge base kernel keeps pulling out them and initiates cascading assertions/retractions as needed. In this case, all the following assertions/retractions, as well as those triggered by the handleUpdateStyleFact function, will be added to the front of the queue in the appropriate order.

In a distributed and shared environment, assertions and retractions to the knowledge base may be invoked by multiple threads (each of which can be an agent component or other agents), and they are likely to be interlaced due to the truth maintenance mechanism or other potential cascading assertions. Under the multi-thread environment, the double-ended queue can be leveraged to manage these kinds of situations. In order to prevent deadlocks and race conditions under multiple invocations, a dedicated thread in the knowledge base kernel was created for processing the queue of pending operations.

4.4.6 Supporting the Semantic Web

For better interoperability and future extensibility, the knowledge base was extended with a mapping mechanism to support semantic Web standard languages.

Specifically, a mapping specification between OWL-Lite [121] and FactType definition of the knowledge base was developed. In addition, a series of translation algorithms for converting rules to SWRL [122][123] and vice versa was also implemented. Using standard languages can enhance the exchangeability and mobility of agent knowledge. Therefore, virtually any editing tools can be chosen and used by knowledge engineers so long as the standard languages can be appropriately supported.

Figure 4.9 illustrates the framework for how the knowledge base can be extended with semantic Web compatibility [124]. In order to be exchangeable with other semantic Web-compatible systems such as third party-editing tools, the knowledge in legacy formats needs to be translated into representations accommodating semantic Web standards. The two core components in this framework are 1) a translator for converting knowledge in legacy formats into semantic Web standards such as OWL and SWRL; and 2) a translator that converts knowledge in the reverse direction. The two translators bridge the knowledge representation gap between the knowledge base and the semantic Web standards. The detailed design and implementation of these two translators are assumed to be system-specific and depend on which semantic Web languages are to be used.

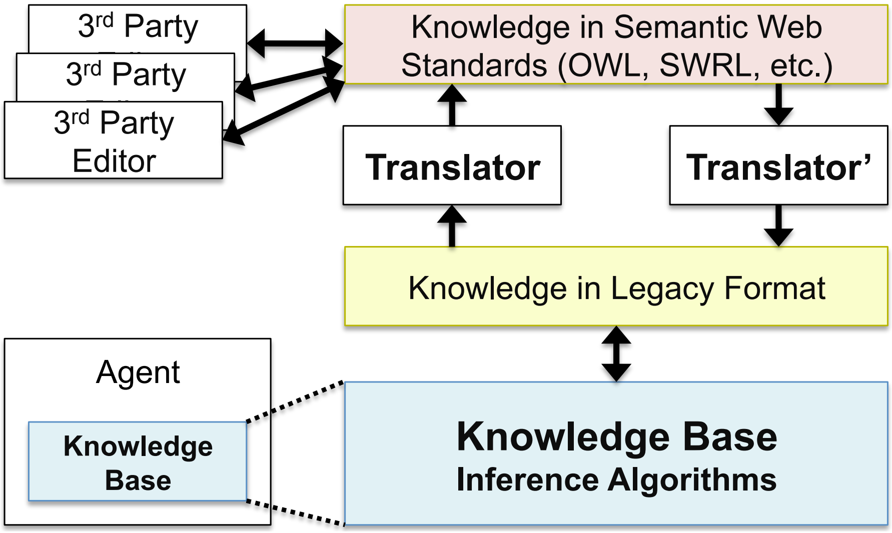


Figure 4.9. The framework for enabling semantic Web compatibility.

4.4.6.1 Preliminaries

The goal of the semantic Web is to enable the contents on the Web to be machine-readable and machine-processable. To reach the goal, XML (Extensible Markup Language)-based knowledge representation languages have been developed for this purpose. The semantic Web layered architecture contains language specifications of increasing expressive power layered on top of lower ones. XML provides the basic syntactical specification regarding how to organize information. On top of XML, RDF (Resource Description Framework) is designed to provide further expressiveness such as subject-predicate-object triples and other complicated data structures using vocabulary for describing classes and concept hierarchies as well as for the concepts of domain and range.

Web Ontology Language (OWL) [121] and its succeeder OWL 2 [125][126] emerged as the standard, derived from previous works such as DAML+OIL rooted in description logics. An OWL-based ontology consists of three major elements: *Classes*, *Properties*, and *Individuals*. A Class is a type definition regarding a concept, which can be organized in a hierarchical structure to represent inheritance relationships. A Property is a binary relationship defined with domains and ranges, usually being used to construct a slot of a class. Two types of properties supported by OWL: a *DatatypeProperty* with a primitive data type as its range such as string or integer defined in XML Schema and 2) an *ObjectProperty* with a class as its range. An Individual is an instantiation of a class based on the class definition and its associated properties.

OWL has three sublanguages with different expressive powers. OWL-Lite supports primary description functions and has some restrictions to ensure computability, such as constraining cardinality to be either 0 or 1. OWL-DL is for situations that need maximum expressiveness without losing computational completeness. OWL-Full is the sublanguage with the maximum expressiveness but without computational guarantees. In this research, we focus on the sublanguage of OWL-Lite, and we leave OWL-DL as the next step in future research.

Due to the restrictions of description logics, some types of knowledge are too complicated to be represented in description logic and are only feasible to be represented in first order logic. These knowledge types are usually represented in rules and being executed in rule-based systems. Semantic Web Rule Language (SWRL)

[122] is one of the standards for rule representation, built on top of RuleML and the OWL language. SWRL enables the representation of knowledge in Horn logic, such that inference knowledge can be realized in rules with antecedents and consequences. RIF (Rule Interchange Format) [127] has also emerged as a standard for representing rule knowledge.

In order to process knowledge encoded in the semantic Web languages, efforts have been made to implement those semantics in rule-based systems [128]. In addition, the interoperability between SWRL and other rule-based systems has also been explored in [129]. However, the issue of bridging and migrating existing rule-based systems to semantic Web standards was not addressed in either work. For other rule languages, SweetJess [130] provides a bi-directional translator between DamlRuleML and Jess [118] rules. However, the research related to this work did not significantly address the details of the algorithms nor the issue of migrating existing rule sets.

In addition to inference, tools for creating and editing knowledge represented in semantic Web standards are also important. A SWRL editor with graphical user interfaces has been developed under the Protégé framework [131]. This editor is integrated with an OWL library and can be utilized to verify the structure and correctness of SWRL rules against the underlying ontology.

4.4.6.2 OWL Elements and FactType/Fact

As noted previously, the three fundamental elements in an OWL ontology are Classes, Properties, and Individuals. A Property can be either a DatatypeProperty or an ObjectProperty associated with a Class, whereas Individuals are the instantiations of a Class. Based on this concept, our approach is to translate a Class along with its Properties into a FactType in the knowledge base, and then translate Individuals of this Class into Facts of this FactType.

Due to the performance consideration and inherited restrictions of the R-CAST knowledge base, the current design is focused on supporting knowledge represented in OWL-Lite and SWRL rules with axioms based on OWL-Lite.

The procedure for translating an OWL-Lite ontology into a knowledge base representation is shown in Algorithm 2. In each resulting FactType, we created an additional property named "InternalID" for storing internally-generated identifiers of individuals instantiated based on its corresponding Class. The InternalID is used as an internal reference in the knowledge base for a Fact to refer to itself.

Algorithm 2 shows the main function of the translation. Lines 2–4 ensure every Class axiom will certainly be translated into a FactType. Since every Class name in an OWL ontology is unique, each Class will be translated into exactly one FactType. Similarly, every Property name is unique, so each Property will be converted into exactly one Property in a FactType. Lines 6–9 and 10–13 deal with DatatypeProperty and ObjectProperty axioms, respectively. The restrictions in lines 6 and 10 ensure only Property axioms having a domain to the current Class will be converted. For Property axioms that have no domain, since there is no associated Class, there will be no Individual using these Properties; therefore it is safe to ignore them. If a Property axiom has a domain, there must exist a corresponding Class axiom; otherwise the OWL document is invalid. Since each Class will be translated into a unique FactType along with its associated Properties, the concepts in the OWL ontology will be preserved in the knowledge base representation.

The procedure for translating FactTypes to an OWL-Lite ontology is illustrated in Algorithm 3. Since the R-CAST knowledge base system does not support cross

Algorithm 2 From Class/Property definitions to FactTypes.

```
Require: A set of Class axioms $C = \{c_1, .., c_m\}$, a set of DatatypeProperty axioms
    $DP = \{dp_1, .., dp_n\}$, and a set of ObjectProperty axioms $OP = \{op_1, .., op_o\}$.
 1: function $OWLtoFactTypes$ ($C$, $DP$, $OP$)
 2: for all $c_i \in C$ do
 3:    $factTypeName \leftarrow$ the name of $c_i$
 4:    Create a FactType $ft$ named $factTypeName$
 5:    $AddPropertyToFactType$ ($ft$, “InternalID”)
 6:    for all $\{dp_j | dp_j \in DP \cap op_j.\text{domain} = c_i\}$ do
 7:       $propertyName \leftarrow$ the name of $dp_j$
 8:       $AddPropertyToFactType$ ($ft$, $propertyName$)
 9:    end for
10:    for all $\{op_j | op_j \in OP \cap op_j.\text{domain} = c_i\}$ do
11:       $propertyName \leftarrow$ the name of $op_j$
12:       $AddPropertyToFactType$ ($ft$, $propertyName$)
13:    end for
14: end for
```

Algorithm 3 From FactTypes to Class/Property definitions.

```
Require: A set of FactType $FT = \{ft_1, .., ft_m\}$
 1: function $FactTypesToOWL\ (FT)$
 2: for all $ft_i \in FT$ do
 3:     $className \leftarrow$ the name of $ft_i$
 4:     Create an OWL Class $c_i$ named $className$
 5:     Get the property list $p_i =< p_{i,1}, .., p_{i,n} >$ from $ft_i$
 6:     for all $p_{i,k} \in p_i$ do
 7:         $propertyName \leftarrow$ the name of $p_{i,k}$
 8:         $uniquePN \leftarrow className.``\_".propertyName$
 9:         Make a DatatypeProperty $dp_{i,k}$ named $uniquePN$
10:         $(dp_{i,k}$.domain$) \leftarrow c_i$
11:         $(dp_{i,k}$.range$) \leftarrow$ String
12:     end for
13: end for
```

reference from one FactType to another, every property of a FactType is translated into a DatatypeProperty.

In Algorithm 3, lines 2–4 ensure every FactType will be translated into a Class axiom. Since every FactType name in the knowledge base is non-duplicated, each FactType will be translated into exact one Class with a unique name. Line 6 ensures all Properties in the FactType will be handled. Line 8 generates a unique name for each OWL Property by appending the Property name to its Class name. Since the Class name is unique, and the Property names under the same FactType are non-duplicated, the name of the OWL Property is unique. Lines 9–11 create a DatatypeProperty for each property of the FactType. Line 10 defines the domain to be its associated Class, and line 11 sets the range to a String. Since each FactType will be translated into a unique Class along with its associated Property definitions, FactType definitions from the knowledge base can be represented in the OWL ontology.

For Individuals and Facts, Algorithm 4 shows the translation algorithm that converts OWL-Lite Individuals into Facts in the knowledge base. Some of the implementation can be realized with hash tables such that the complexity can be reduced. When creating a Fact, the URI (Uniform Resource Identifier) of the Individual will be used as its InternalID since each URI is unique in an OWL ontology. If there is no URI designated to an Individual, a unique identifier will be generated by invoking the function *createAGlobalUniqueID*(). This function is designed for generating a non-duplicated string in the system. To illustrate the usage of InternalIDs, Figure 4.10 shows an example with two classes, where one is referred by the other through an ObjectProperty. The two FactTypes "Build-

Algorithm 4 From OWL-Lite individuals to Facts.

```
Require: A set of FactType $FT = \{ft_1, ..., ft_m\}$
    A set of Individual $D = \{d_1, ..., d_n\}$, where each of $d_i$ is created according to a
    class definition and its associated property definitions.
function $IndividualtoFacts$ ($FT$, $D$)
for all $d_i \in D$ do
    $c \leftarrow$ the class of $d_i$
    $ft \leftarrow$ the corresponding FactType of $c$
    Create a Fact $f$
    $id \leftarrow$ the URI of $d_i$
    if $id = null$ then
        $id \leftarrow$ createAGlobalUniqueID()
    end if
    $SetPropertyValueToFact$ ($f$, "InternalID", uri)
    for all $d_i$'s property of $p_{i,j}$ do
        $pName \leftarrow$ the name of $p_{i,j}$
        $value \leftarrow$ the value of $p_{i,j}$
        $SetPropertyValueInFact$ ($f$, $pName$, $value$)
    end for
end for
end function

function $SetPropertyValueInFact$
    ($f : Fact$, $propertyName : String$, $value : String$)
$ft \leftarrow$ the FactType of $f$
Retrieve the property list $p =< p_1, ..., p_n >$ from $ft$
Retrieve the value list $v =< v_1, ..., v_n >$ from $f$
for all $p_i \in p$ do
    if $p_i = propertyName$ then
        $v_i \leftarrow value$
        update the value list $v$ in $f$
        return
    end if
end for
end function
```

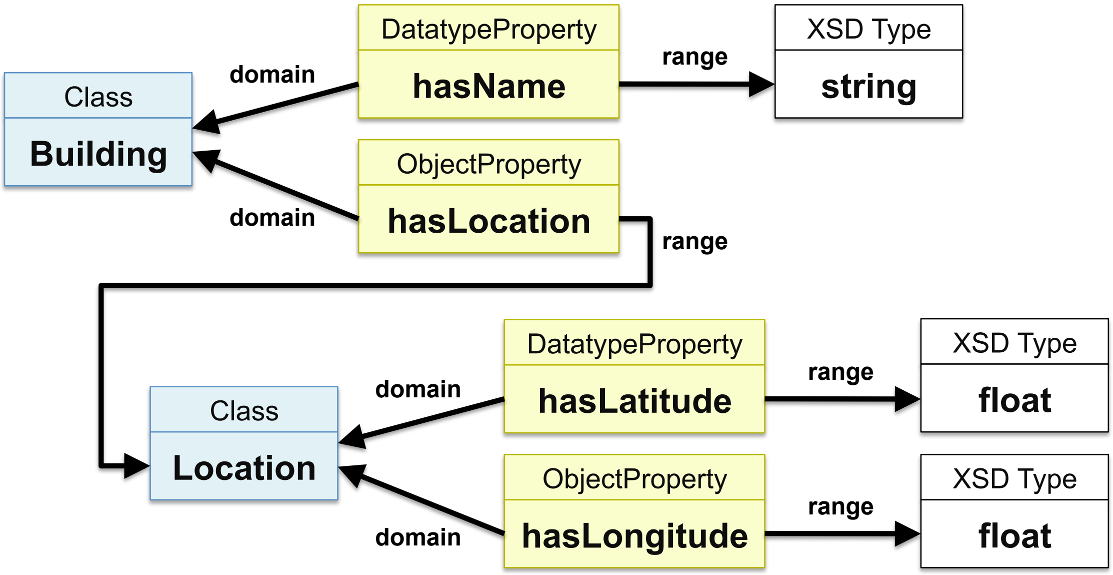


Figure 4.10. An example ontology in OWL-Lite.

ing" and "Location" listed below are generated by the translation algorithm. The last property "hasLocation" of the Building FactType is from the ObjectProperty "hasLocation." In the Building Fact, its "hasLocation" property is set to the reference (i.e., the InternalID) of the Location Fact.

For the translation from Facts to OWL-Lite Individuals, as shown in Algorithm 5, it is straightforward to create an OWL Individual and then assign all the property values by using its associated OWL Property definitions.

```
(FactType Building (?IID ?hasName ?hasLocation))
(FactType Location (?IID ?latitude ?longitude))
(Fact Building (id0001  BuildingA  id0002))
(Fact Location (id0002  45  135))
```

4.4.6.3 Rules

A SWRL rule is encoded in the *Imp* data structure, which is composed of a body element for the antecedents and a head element for the consequence. The content of a head or a body is an *AtomList*, which is composed of a number of *Atoms*. There are four primary types of Atoms in SWRL: *ClassAtom*, *DatavaluedPropertyAtom*, *IndividualPropertyAtom*, and *BuiltinAtom*. ClassAtom is a unary predicate

Algorithm 5 From Facts to OWL-Lite individuals.

```
Require: A set of Facts $F = \{f_1, ..., f_m\}$
function $FactsToOWL$ $(F)$
for all $f_i \in F$ do
    $ft_i \leftarrow$ the FactType of $f_i$
    Get $ft_i$'s corresponding OWL Class $c_i$
    $iName \leftarrow$ the ID of $f_i$
    Make an OWL Individual $d_i$ of class $c_i$, named $iName$
    Get the property list $p_i =< p_{i,1}, ..., p_{i,n} >$ from $ft_i$
    Get the value list $v_i =< v_{i,1}, ..., v_{i,n} >$ from $f_i$
    for all $p_{i,k} \in p_i$ do
        Get $p_{i,k}$'s corresponding OWL Property $PR_{i,k}$
        $d_i.PR_{i,k} \leftarrow v_{i,k}$
    end for
end for
end function
```

for declaring the class of an object, where the object can be either an identifier or a variable. DatavaluedPropertyAtom and IndividualPropertyAtom are binary predicates that are used to associate an object with a value or another object. A BuiltinAtom is an element embedded with a functional predicate that supports a predefined operation. There are a number of built-in operations defined in the SWRL ontology, and users are also allowed to create custom operations.

According to the SWRL definition, there are only minimum restrictions on how a rule should be defined in SWRL; hence rules in SWRL can be encoded in a very flexible way, but challenges are introduced when mapping them to rules of existing forward-chaining rule-based systems with inherited restrictions. In the current implementation of our system, we support a complete mapping from the R-CAST rule-based system to SWRL, but only a partial mapping from SWRL to the system of R-CAST.

Referring to Figure 4.5 that shows the syntax of rules of the R-CAST knowledge base, an antecedent can be either a predicate based on a FactType definition or a functional predicate which implements an operation. The procedure for translating a R-CAST rule set to SWRL rules is outlined in Algorithm 6. This algorithm consists of three major parts. The first part (lines 7–13) translates antecedent predicates based on FactType definitions, or non-functional predicates, into SWRL

ClassAtoms and PropertyAtoms. For example, the first antecedent of the example rule in 4.4.1 “(Person (?name ?age male))” will be translated into four Atoms as follows. (Note: The “#” sign is for indicating that it is an URI that can be looked up in the ontology, where the namespace is ignored.)

```
ClassAtom(#C1, #Person)
DatavaluedPropertyAtom(#C1, #name)
DatavaluedPropertyAtom(#C1, #age)
DatavaluedPropertyAtom(#C1, "male")
```

The second part (lines 14–20) translates functional predicates in the antecedent such as arithmetic and comparison operators. It creates a BuiltinAtom with its built-in element referring to the URI of the corresponding SWRL built-in ontology. The arguments in a BuiltinAtom comprise an RDF List, and the sequence of the arguments should be carefully organized if the parameters are defined differently between the functional predicate and the SWRL built-in. For example, the second antecedent of the example rule in 4.4.1 “($<$ (?age12))” will be translated into a BuiltinAtom as follows:

```
BuiltinAtom(builtin=swrlb:lessThan,
            arguments=(#age, 12))
```

The last part (lines 21–31) of the algorithm translates the consequence into a SWRL ClassAtom and several PropertyAtoms. For the consequence, two options must be considered. It is necessary to identify whether a rule is designed 1) to update existing individuals or 2) to create new ones. If the rule is designed to create new individuals, we need to prepare a new SWRL variable in the consequence. However, SWRL was originally designed to support updating existing individuals by asserting new relationships or properties; thus there is no standard syntax for creating new variables representing newly-created individuals in the consequence. In addition, SWRL disallows unbound variables in the consequence, and all variables should be declared in the antecedents. Therefore, in order to support creating a new individual, the SWRL built-in extension $CreateOWLThing$ [132] provided by the Protégé group was adopted for our approach. $CreateOWLThing$ is a built-in to be used in the antecedent for declaring a variable to represent the

Algorithm 6 From R-CAST rules to SWRL rules.

```
Require: A set of Rules $R = \{r_1, ..., r_m\}$, where each $r_i = \{P_i, q_i, X_i\}$ is composed of one or more ordered
    antecedents $P_i = \{p_{i,1}, ..., p_{i,n}\}$, a consequence $q_i$, and a set of shared variable $X_i = \{x_{i,1}, ..., x_{i,o}\}$ among
    $p_{i,1}$, ..., $p_{i,n}$, and $q_i$.
function $RulesToSWRL$ ($R$)
for all $r_i \in R$ do
    Create a SWRL Imp object $imp_i$
    for all $x_{i,j} \in X_i$ do
        Create a SWRL Variable named $x_{i,j}$
    end for
    //Non-functional predicates in the antecedent:
    for all $p_{i,k} \in P_i \wedge \neg(p_{i,k}$ is a functional predicate$)$ do
        Get $p_{i,k}$'s corresponding OWL Class $C$
        Create a SWRL Variable named $c_k$
        Make a SWRL ClassAtom($C$, $c_k$) in $imp_i$.body
        $CreatePropertyAtoms$($p_{i,k}$, $c_k$, $X_i$) in $imp_i$.body
    end for
    //Functional predicates in the antecedent:
    for all $p_{i,k} \in P_i \wedge (p_{i,k}$ is a functional predicate$)$ do
        Create a RDF List $args$
        $op \leftarrow$ operator of $p_{i,k}$
        $args \leftarrow$ arguments of $p_{i,k}$
        Create a BuiltinAtom($op$, $args$) in $imp_i$.body
    end for
    //The consequence:
    if $r_i$ is designed to create new facts then
        Create a SWRL Variable named $c_{new}$
        Create a BuiltinAtom("CreateOWLThing", $c_{new}$) in $imp_i$.body
        $c_{head} \leftarrow c_{new}$
    else
        $c_{head} \leftarrow$ the SWRL Variable that represents the individual to update
    end if
    Get $q_i$'s corresponding OWL Class $C$
    Create a SWRL ClassAtom($C$, $c_{head}$) in $imp_i$.head
    $CreatePropertyAtoms$ ($q_i$, $c_{head}$, $X_i$) in $imp_i$.head
end for
end function

function $CreatePropertyAtoms$
($p$ : $predicate$, $c$ : a SWRL Variable designated for this predicate, $X$ : the variable set of the rule)
for all arguments $a_i$ of $p$ do
    Get $a_i$'s corresponding OWL Property $PR$
    $arg1 \leftarrow c$
    if $a_i \in X$ then
        $arg2 \leftarrow$ URI of corresponding SWRL Variable
    else
        $arg2 \leftarrow$ value of $a_i$
    end if
    if $PR$ is a DatatypeProperty then
        Create a DatavaluedPropertyAtom($PR$, $arg1$, $arg2$)
    else
        Create a IndividualPropertyAtom($PR$, $arg1$, $arg2$)
    end if
end for
```

Algorithm 7 From SWRL rules to R-CAST rules.

```
Require: A set of SWRL Rules $SR = \{sr_1, ..., sr_m\}$, where each $sr_i = \{Head_i, Body_i, V_i\}$ is composed of
    a body $Body_i = \{CA_i, PA_i, BI_i\}$ consisting of ClassAtoms, PropertyAtoms, and BuiltinAtoms; a head
    $Head_i = \{ca'_i, PA'_i\}$ consisting of only a ClassAtoms and some PropertyAtoms; and a set of Variables $v_i$.
function $SWRLToRules$ ($SR$)
for all $sr_i \in SR$ do
    Create a rule $r_i$
    //Non-functional predicates in the antecedent:
    for all $ca_{i,j} \in CA_i$ of $Body_i$ do
        $C_{i,j} \leftarrow$ the OWL Class enclosed in $ca_{i,j}$
        $ft_{i,j} \leftarrow$ the corresponding FactType of $C_{i,j}$
        $c_{i,j} \leftarrow ca_{i,j}$.argument1
        Create a Predicate $pr_{i,j}$ based on $ft_{i,j}$'s definition
        for all $pa_{i,k} \in PA_i$ of $Body_i$ do
            $arg1 \leftarrow pa_{i,k}$.argument1
            if $arg1 = c_{i,j}$ then
                $pName \leftarrow$ the Property name enclosed $pa_{i,k}$
                $arg2 \leftarrow pa_{i,k}$.argument2
                $SetPropertyValueInPredicate(pr_{i,j}, pName, arg2)$
            end if
        end for
        $r_i$.antecedent = $r_i$.antecedent + $pr_{i,j}$
    end for
    //Functional predicates in the antecedent:
    for all $bi_{i,l} \in BI_i$ do
        $op \leftarrow$ the operator enclosed in $bi_{i,l}$
        $args \leftarrow$ arguments of $bi_{i,l}$
        Create a Functional Predicate $fpr_{i,l} = (op, args)$
        $r_i$.antecedent = $r_i$.antecedent + $fpr_{i,l}$
    end for
    //The consequence:
    $C'_i \leftarrow$ the OWL Class enclosed in $ca'_i$
    $ft'_i \leftarrow$ the corresponding FactType of $C'_i$
    $c'_i \leftarrow ca'_i$.argument1
    Create a Predicate $pr'_i$ based on $ft'_i$'s definition
    for all $pa'_{i,k} \in PA'_i$ of $Head_i$ do
        $arg1 \leftarrow pa'_{i,k}$.argument1
        if $arg1 = c'_i$ then
            $pName \leftarrow$ the Property name enclosed $pa'_{i,k}$
            $arg2 \leftarrow pa'_{i,k}$.argument2
            $SetPropertyValueInPredicate(pr'_i, pName, arg2)$
        end if
    end for
    $r_i$.consequence = $pr'_{i,j}$
end for
end function

function $SetPropertyValueInPredicate$
    ($pr : Predicate, propertyName : String, value : String$)
$ft \leftarrow$ the FactType of $pr$
Retrieve the property list $p =< p_1, ..., p_n >$ from $ft$
Retrieve the value list $v =< v_1, ..., v_n >$ from $pr$
for all $p_i \in p$ do
    if $p_i = propertyName$ then
        $v_i \leftarrow value$
        update the value list $v$ in $pr$
        return
    end if
end for
end function
```

newly created individual, such that in the consequence we can refer to the new individual using this variable.

With a similar structure, the translation algorithm from SWRL to R-CAST rules is shown in Algorithm 7. It is a straightforward to reverse the way of mapping in each of these steps. However, it is worth mentioning that not all SWRL rules can be translated into R-CAST rules. Since we have the assumption that the ontology is in OWL-Lite, the SWRL rules are also supposed to only reason about OWL-Lite individuals. Due to this restriction, each object declared with ClassAtom should have only non-duplicated property atoms (DatavaluedPropertyAtom or IndividualPropertyAtom) in order to satisfy the cardinality requirement of OWL-Lite.

4.5 Other Decision Modules

To enhance cyber situation recognition, we also incorporated two other decision modules into our implementation. These can handle more complicated decision requirements and provide alternative ways to support decision making. The first module is inspired by the recognition-primed decision (RPD) model, which is a naturalistic decision model developed by Klein [14][15] and has been widely adopted in military and firefighter training. A cognitive model, it explains how a human expert makes decisions under time-stress—by matching the current situation against one's experiences, followed by mental simulation and evaluation. The second module is based on evaluating and ranking a set of options based on a weighted-sum approach.

4.5.1 A Computational RPD Module

In the RPD model, recognition is based on cues associated with experiences. The recognition procedure is similar to case matching in case-based reasoning [133]. In our implementation, cues are a list of conditions represented as predicates in the knowledge base. Cue matching is implemented as a query to the knowledge base to check whether the cue is true in the current situation. The cues of an experience can be defined as the primary decision factors elicited from the domain experts.

In the RPD model, the behavior of humans making decisions under emergent situations is rational but not optimal. Instead of thoroughly retrieving all previous experiences and considering all possible solutions, human experts tend to make a satisficing decision—that is, one which is good enough but not necessarily the best one. This is a basic assumption in the RPD model. Since a computer-based agent does not have this limitation, we can leverage the computational resources to enhance our work. Instead of finding only the first-matched experience, the decision module is designed to take advantage of the computational power to consider as many as experiences as possible (ideally the entire set of experiences in the system) to boost the quality of decision.

Inspired by the RPD model, the decision module was designed and implemented as follows. For a given domain, we constructed an experience base which contained experience knowledge elicited from the domain experts. A set of experience instances comprised the experience base. Each experience instance had several cues for matching the current situation. A cue could be either optional or required. Each experience was also associated with an action, which was the default action if there was a match. The action was treated as a decision option for the decision module.

In an experience base, the common features of a subset of experience instances can be abstracted to form experience spaces. Therefore, we can organize the experience base in a tree structure. An experience space can have multiple experience instances which inherit some cues from it. Using the tree structure, the domain experience knowledge can be well organized.

The decision module generates recommendations by matching experience base against the current situation. In the computational environment, the original RPD model is modified to consider all experience instances in the experience base. At run-time, the experience base is traversed by the decision module. All the cues, including both required and optional, are consolidated into information requirements for obtaining the information. The decision module ranks all the experiences according to their similarities, and the one with the highest similarity is reported as the recommendation. The similarity scores and the ranks are being updated whenever there comes new information, such that the user can be up to date with the latest situation and available recommendations.

4.5.2 A Utility-Based Decision Module

The other type of decision module is based on the utility theory [53][54] and a utility-based approach [134]. This module is based on the cost-benefit analysis of a list of options and generates recommendations according to their scores.

For each option, the decision module keeps a list of criteria with their corresponding weights. Each of the criteria is evaluated and assigned with a score, where the higher score is better. These scores will be normalized to a number between 0 and 1, and then ,summed up according to their weights to generate an overall score. This decision module compares the overall scores of all options and displays a ranked list to the user.

This utility-based decision module is implemented in a flexible way such that a criteria value can be either numeric or qualitative. A numeric criteria value can be directly used as the evaluation score or be converted into a different value if the mapping is non-linear or with some constraints such as a cap. Qualitative criteria values need to be converted to a numeric value to fit the weighted-sum evaluation scheme (e.g., mapping qualitative values “Low”, “Medium” or “High” into specific numeric values). This kind of conversion definition is domain-specific and depends on how the users interpret it.

We found that this approach is useful for decision problems where the criteria present a clear view regarding what is good and what is bad. If it is good, a higher score that implies a better utility shall be assigned. Since there is a norm and a clear definition for measuring utility, the utility-based decision module is preferable for dealing with problems that require normalized evaluation such as asset allocation.

4.6 Agent Management Services

The readiness and accessibility of a decision support system is very important. The R-CAST multi-agent architecture was originally designed as a Java application running on a local machine. To improve its readiness and make the system accessible to multiple users from virtually anywhere, we also developed a framework that can encapsulate a group of agents as a service.

Service-oriented computing is a trend in enterprise information infrastructure. Multi-agent systems have the flexibility of information fusion and reasoning capability. Within the scope of a specifically designed agent program, an agent system can behave as configured by the developer and knowledge engineer. However, it is challenging to put a dynamic agent system under the conditions of a service-oriented environment. This section describes a framework which can be used to encapsulate a multi-agent system as a Web service, so that teams of agents can be dynamically configured and instantiated based on users' requests.

With service-based multi-agent systems, cloud computing supported by a team of user-configurable agents becomes feasible. Customers could be enabled to construct agent teams that reside on the server and then assign tasks to these agents. These agents can also be pre-configured by experts to provide specific functionalities such as knowledge inference, decision making, etc. They can also be retrieved and instantiated to perform their designated tasks. In sum, the vision of the agent management services is to enable users to dynamically create and configure a desired group of agents for problem solving.

There has been some research conducted on the relationship between multi-agent systems and service-oriented computing. In [135], the authors pointed out potential research directions for service-oriented multi-agent systems. The categorization of integration approaches between these two areas has also been discussed in [136]. Among those categories, our approach is closest to the one which views

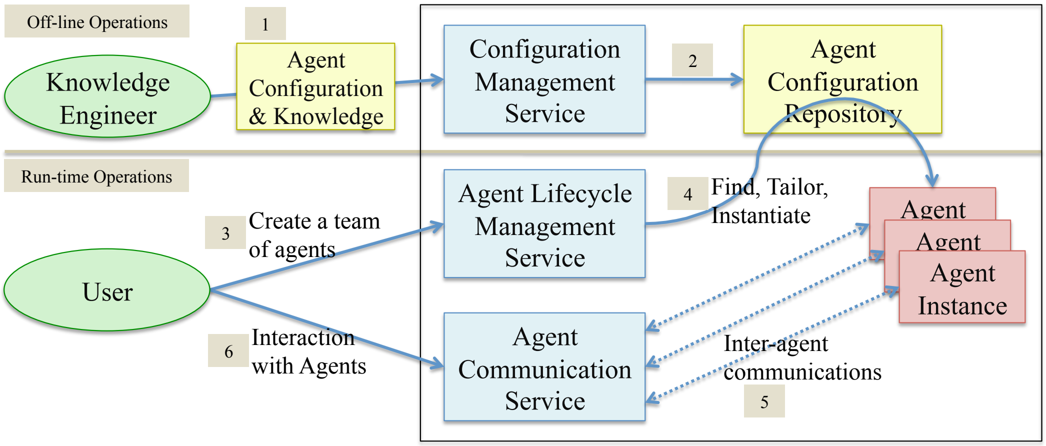

Off-line Operations
1
Knowledge Engineer
Agent Configuration & Knowledge
Configuration Management Service
2
Agent Configuration Repository
Run-time Operations
Agent Lifecycle Management Service
4
Find, Tailor, Instantiate
Agent
Agent
Agent Instance
3
Create a team of agents
User
6
Interaction with Agents
Agent Communication Service
Inter-agent communications
5

Figure 4.11. The architecture of the agent management services.

Web services are provided by agents. As for the dynamic nature of user-defined agent configuration, the concept of dynamic service generation (DSG) mentioned in [136] can be used to support the idea, with the difference that our work is focused on providing multi-agent systems as services.

Figure 4.11 is an overview of the agent management services. There are three major facilitating services supporting this system. The Configuration Management Service is responsible for managing agent configurations. It is linked with the Agent Configuration Repository and provides interfaces for knowledge engineers to store and retrieve agent configurations/knowledge.

The Agent Lifecycle Management Service (ALMS) is responsible for tailoring and instantiating a team of agents based on users' requests. Figure 4.12 is a sequence diagram showing the steps of how a team of agents is created and terminated. To create a team of agents, the user first invokes the *createAgentTeam* operation and specifies which set of agent configurations/knowledge to use (event 1

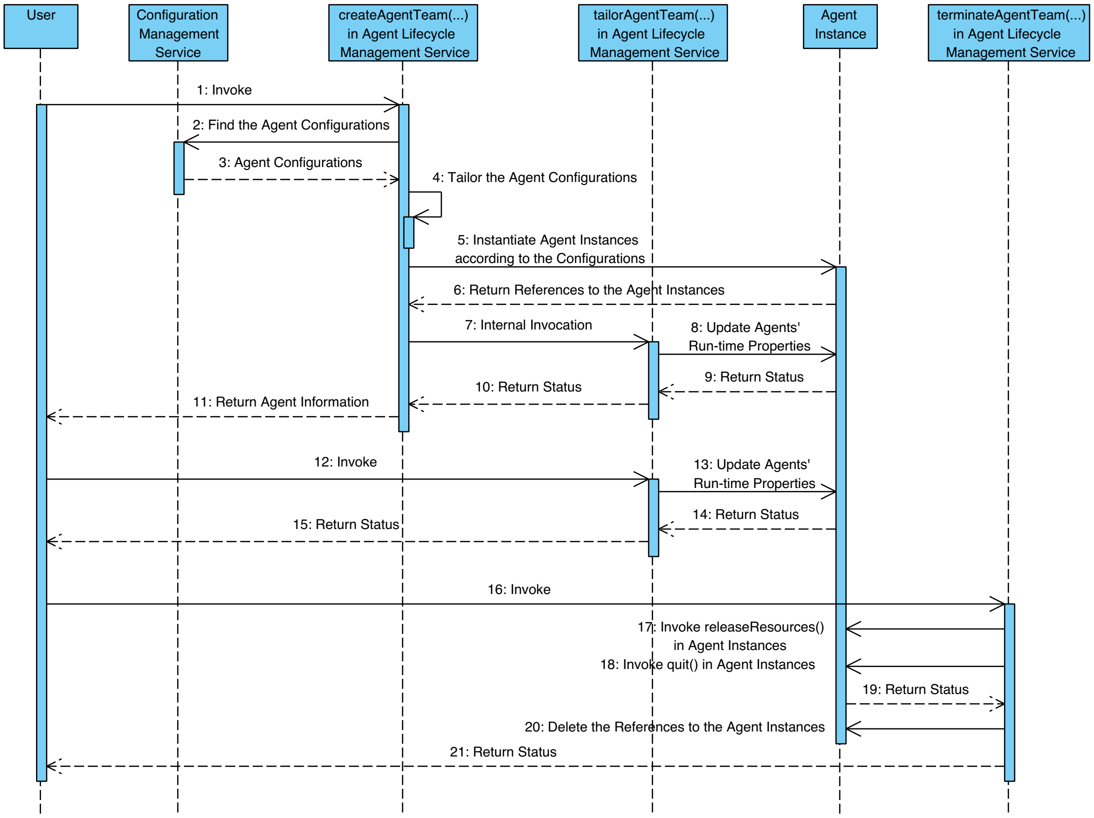

User
Configuration Management Service
createAgentTeam(...) in Agent Lifecycle Management Service
tailorAgentTeam(...) in Agent Lifecycle Management Service
Agent Instance
terminateAgentTeam(...) in Agent Lifecycle Management Service
1: Invoke
2: Find the Agent Configurations
3: Agent Configurations
4: Tailor the Agent Configurations
5: Instantiate Agent Instances according to the Configurations
6: Return References to the Agent Instances
7: Internal Invocation
8: Update Agents' Run-time Properties
9: Return Status
10: Return Status
11: Return Agent Information
12: Invoke
13: Update Agents' Run-time Properties
14: Return Status
15: Return Status
16: Invoke
17: Invoke releaseResources() in Agent Instances
18: Invoke quit() in Agent Instances
19: Return Status
20: Delete the References to the Agent Instances
21: Return Status

Figure 4.12. Sequence diagram of the agent lifecycle management service.

in Figure 4.12). Second, the ALMS retrieves the agent configurations and then tailors the configurations according to the current context (events 2-4). Third, based on these tailored configurations, the ALMS instantiates a team of the agents, keeps their references, and assigns a unique ID to each of the agents (events 5-6). Fourth, the ALMS invokes the *tailorAgentTeam* operation to perform further tailoring needed at run-time (events 7-10). This dynamic tailoring can also be invoked by the user at any time (events 12-15). Last, information about the team of agents is returned to the user (event 11). After the agents finish their tasks, the user can invoke the *terminateAgentTeam* operation, such that the ALMS will destroy these agent instances and release resources (events 16-21).

The Agent Communication Service facilitates agent-to-agent communications by providing a central storage for agents to exchange their messages. As illustrated in Figure 4.13, every registered agent has its own mailbox. Since each agent has a unique ID assigned by the ALMS, there is no confusion about the owner of the each mailbox. While sending a message, the sender first encodes its own ID and the recipient's ID in the header of the message and then invokes the *sendMessage* operation. Each agent should regularly invoke the *retrieveMessage* operation to check whether it has incoming messages. In addition, the end user can also directly communicate with the agents through the Agent Communication Service.

These three facilitating services work together to support agent management operations. As illustrated with the numbers in Figure 4.11, the Configuration Management Service takes agent configurations and then stores them in the Agent Configuration Repository. At run-time, the Agent Lifecycle Management Service

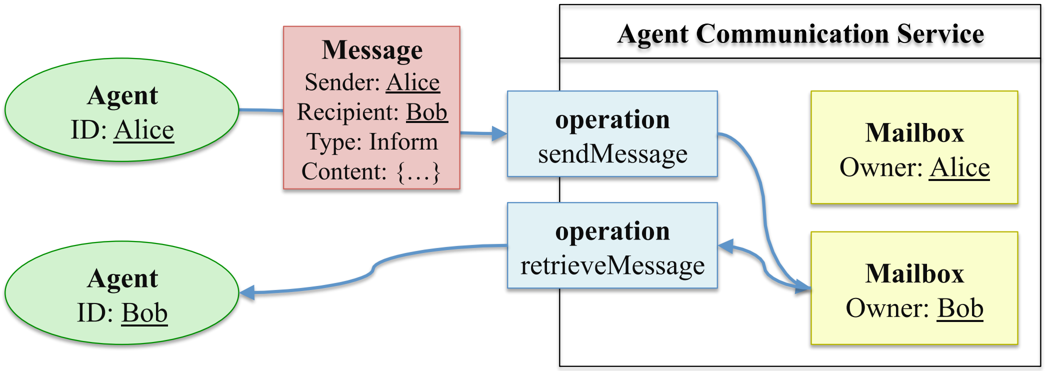


Figure 4.13. The agent communication service.

accepts users' requests and instantiates agent instances. These agent instances work together as a team to perform their designated tasks and communicate with each other or the end users through the Agent Communication Service.

Chapter 5

Simulations and Experiments

In order to validate the idea of experience-based cyber situation recognition, we performed an experiment using the agent-based system described in Chapter 4, incorporating the element of experience relaxation. We developed case studies to demonstrate the strengths and limitations of our approach.

5.1 Experiments on Experience Relaxation

The validation of experience relaxation is based on a simulation model which can produce a large amount of run-time intrusion sequences to cover as many potential cases as possible. We present the simulation models and the results in the following sub-sections.

5.1.1 The Alert Sequence Simulator

Figure 5.1 is an illustration of the simulation model used to produce alert sequences. An alert sequence is generated as follows.

- First, the simulator reads a predefined attack scenario which is represented as an ordered list of attacking steps.

- Second, beginning at the first attacking step, the simulator generates a time stamp for an event and then randomly produces a time interval based on a statistical distribution to determine the time stamp for the subsequent attacking step.

- Third, the simulator generates all the alerts for a given event as well as the delay periods for the alerts. The time stamp for each alert is calculated by adding the delay to the time stamp of its triggering event. When producing each alert, a random number generator is consulted to decide whether or not it should be marked as a missing alert. A missing alert will not be made available to the user.
- In the meantime, an alert correlation engine works simultaneously to produce correlation information.
- Finally, the alerts and their correlation sequences will be sent to the receiver (e.g., the experience generator or the Recognizer) according to their time stamps in ascending order. It is worth mentioning that the information about the triggering event becomes a part of the alert it triggers. Since events are hidden from the user, an event will not be noticed by the user until one of its associated alerts is successfully delivered to the user.

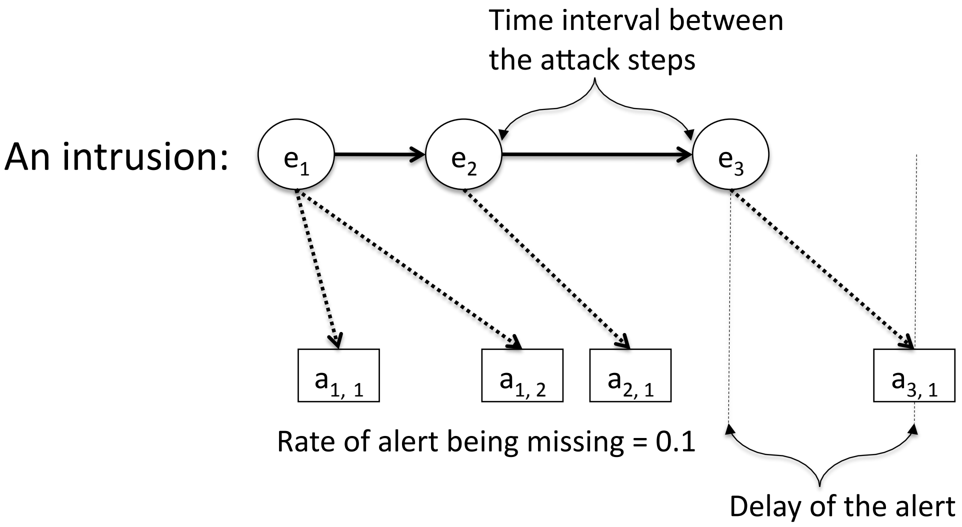


Figure 5.1. The simulation model.

5.1.2 Experimental Settings

Based on the simulation model, we used the technique of discrete-event system simulation [137] to perform a series of simulated experiments. The emphases of our experiments are listed as follows:

- to simulate the experience capturing and accumulation using a systematic approach;
- to perform automatic experience relaxation and see how experience relaxation improves situation recognition;
- to check the usefulness of the experience-based approach using these captured and relaxed experiences.

Figure 5.2 outlines the key actions, the components, and the flow of the experiment, which is designed as a two-phase process. The first phase is to capture

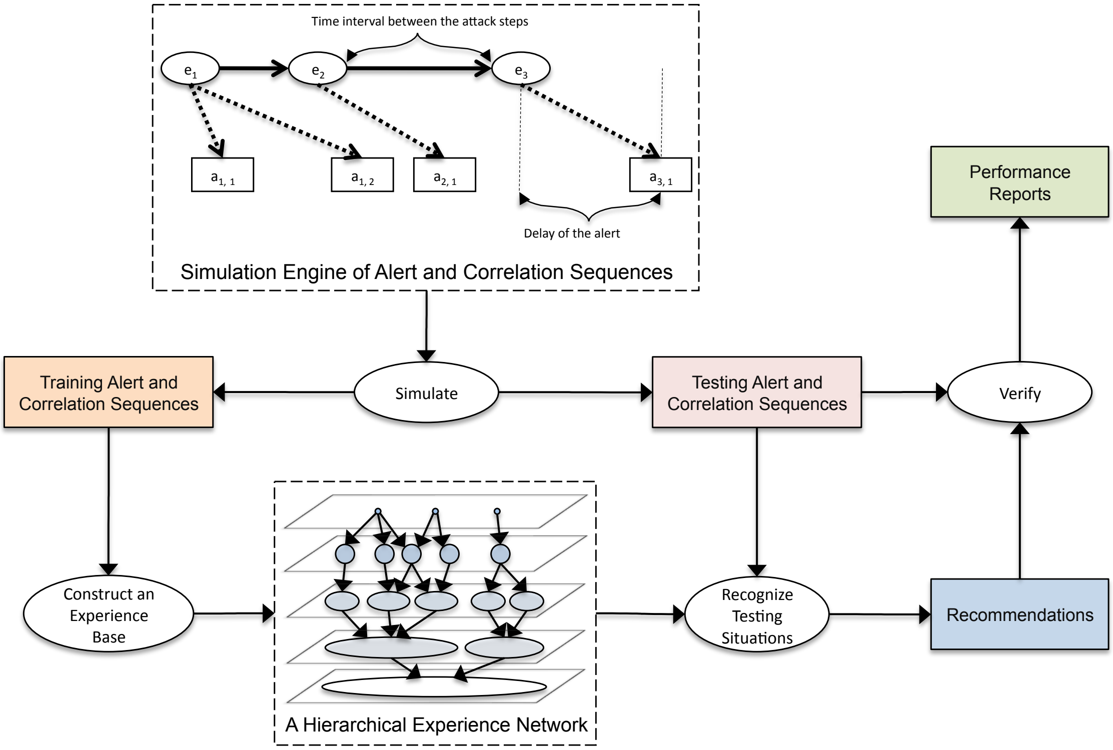

Time interval between the attack steps
e_1
e_2
e_3
$a_{1,1}$
$a_{1,2}$
$a_{2,1}$
$a_{3,1}$
Delay of the alert
Simulation Engine of Alert and Correlation Sequences
Performance Reports
Training Alert and Correlation Sequences
Simulate
Testing Alert and Correlation Sequences
Verify
Construct an Experience Base
A Hierarchical Experience Network
Recognize Testing Situations
Recommendations

Figure 5.2. Experimental design.

experiences, relax experiences, and integrate newly captured and relaxed experiences into the hierarchical experience network. The second phase is to perform situation recognition by feeding simulated testing alert and correlation sequences into the agent; the recommendation generated by the agent will be compared against the ground truth in order to determine whether the recognition is correct or not.

The experiments were performed using the scenario from [49], involving three hosts as shown in Figure 5.3 and two similar attacking approaches as illustrated in Figure 5.4. We assumed the rate of missing alert to be 10%, such that approximately 10% of the alerts will not be delivered. We also assumed that the alert correlation engine could successfully report 50% of the correlated alerts.

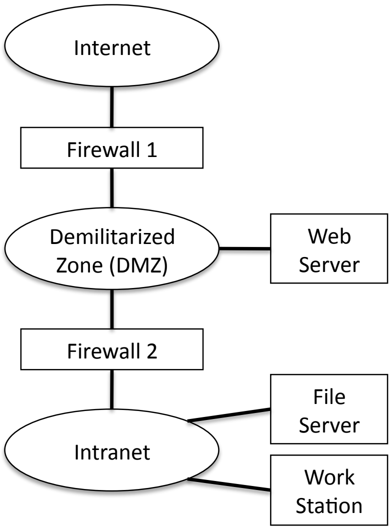


Figure 5.3. An example network consisting of three hosts.

Based on the experimental setting, we repeated the experiment 100 times. In each session we ran the agent using different quantities of experiences in an accumulating fashion. We started with an agent having five Level 0 experiences and their relaxed versions. Then, we began integrating five more Level 0 experiences and their relaxed versions into the same experience base through the cumulative process illustrated in Figure 3.18. Each time, we performed situation recognition on the same set of testing alert and correlation sequences. For each session, we collected the percentage of correct recognitions based on the verification between agent recommendations and the ground truths. Note that the simulation was

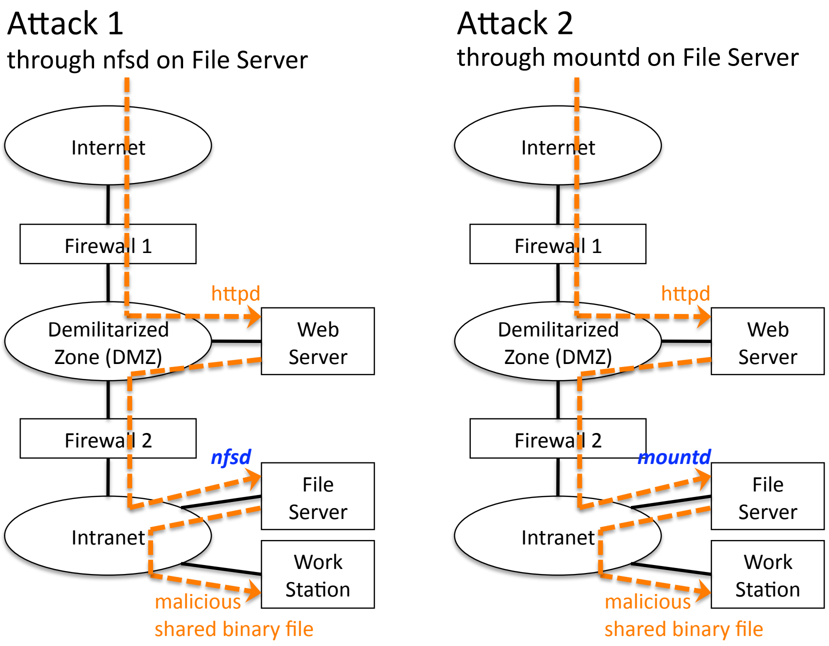


Figure 5.4. Attack scenarios on the example network.

designed in a way making parallel computing applicable, so that the experiments could be performed on compatible platforms.

5.1.3 Results and Discussions

Figure 5.5 shows the recognition results performed by agents with only experiences of Level 1 relaxation. The rate of correct recognitions increases as the number of experiences is accumulated. However, even with 100 sets of Level 1 experiences, the rate of correct recognition is only 48% in average. Analysis suggests this occurred because the experience base failed to recognize situations with limited coverage, which will be further examined in a following section. As discussed in Chapter 3, coverage will be extended if the process of experience relaxation can be applied.

In order to test whether experience relaxation can make a difference, additional experiments using additional Level 2 and Level 3 experiences were performed given the same set of testing situations (i.e., the same set of testing alert and correlation sequences).

Figure 5.6 shows the recognition results made by agents with experiences from Level 1 and Level 2 relaxation, and Figure 5.7 shows the results by agents with

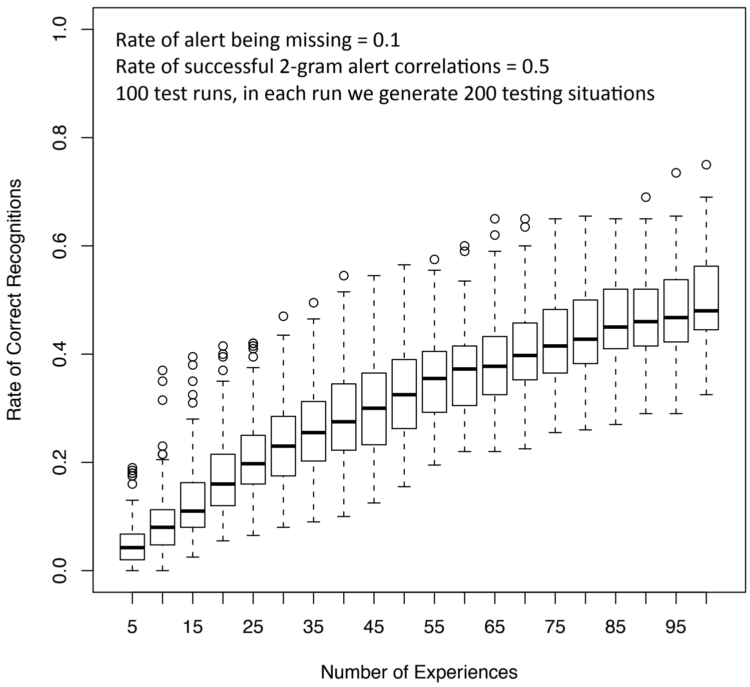


Figure 5.5. Situation recognition using Level 1 experiences.

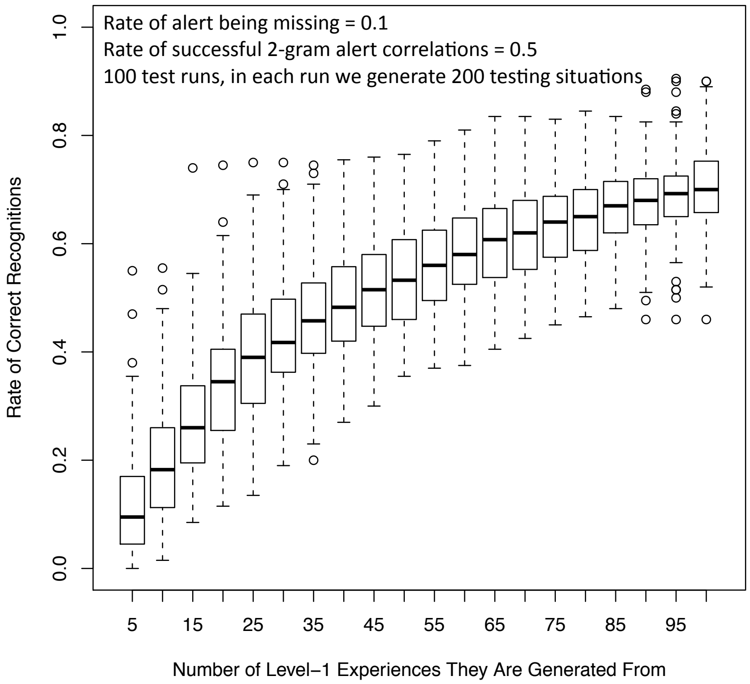


Figure 5.6. Situation recognition using Level 1 and Level 2 experiences.

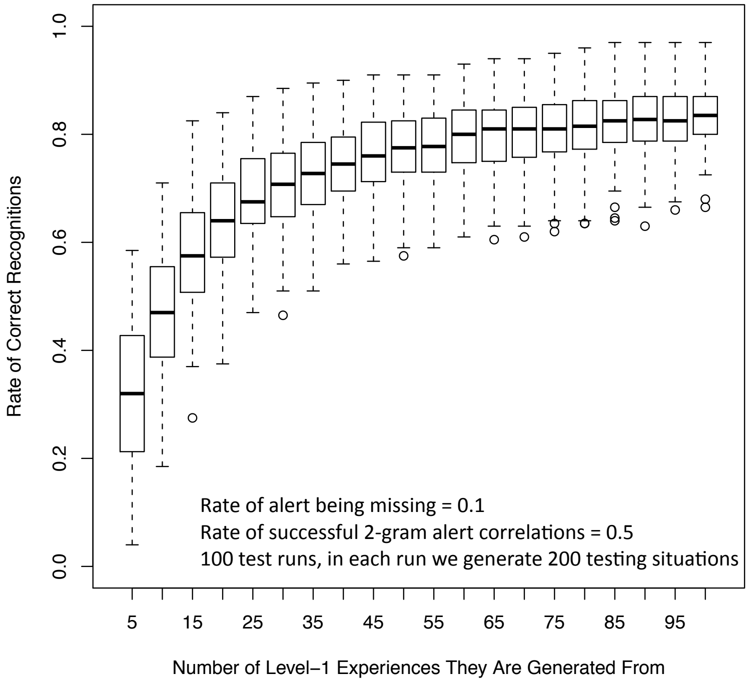


Figure 5.7. Situation recognition using Level 1, Level 2, and Level 3 experiences.

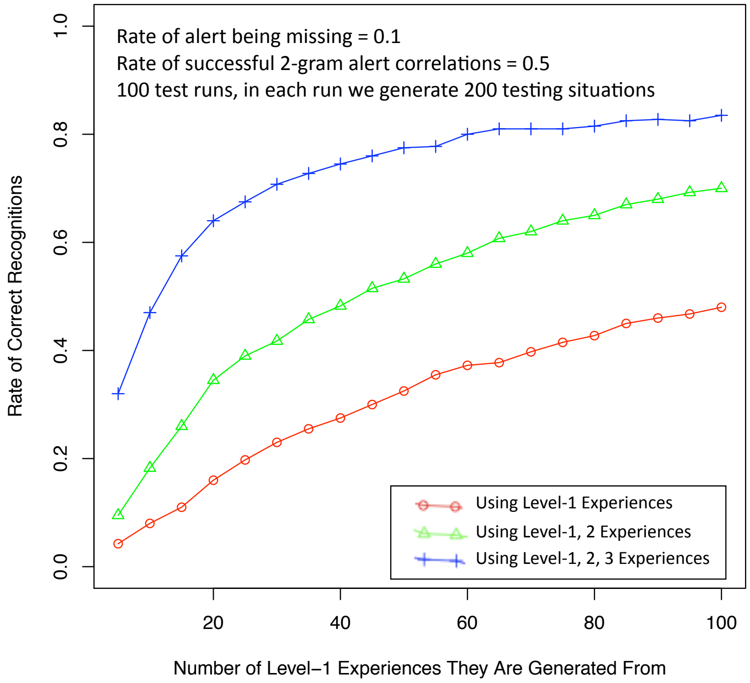


Figure 5.8. Rate of correct situation recognitions.

Table 5.1. p-values from the Student's t-tests on the rates of correct recognitions using different levels of experience relaxation.

		p-values	
		Between L1 experiences and L1+L2 experiences	Between L1+L2 experiences and L1+L2+L3 experiences
Number of level-1 experiences they are generated from	5	8.22E-09	1.69E-22
	10	1.15E-13	3.15E-40
	15	1.78E-19	2.15E-48
	20	5.28E-25	6.68E-50
	25	5.71E-27	3.87E-49
	30	2.73E-30	7.57E-48
	35	2.08E-32	2.89E-47
	40	1.30E-32	1.52E-45
	45	1.28E-34	2.21E-43
	50	5.65E-37	1.74E-42
	55	2.03E-38	2.21E-40
	60	9.29E-41	7.27E-41
	65	1.12E-39	6.61E-38
	70	1.12E-39	2.71E-36
	75	2.32E-40	2.55E-35
	80	1.37E-40	2.58E-33
	85	1.06E-43	8.83E-33
	90	8.56E-43	3.62E-31
	95	7.12E-41	1.65E-28
	100	9.50E-41	5.40E-28

experiences from Level 1, Level 2, and Level 3 relaxation. Figure 5.8 integrates the mean rate of correct recognition from each experiment into one diagram. The rates of correct recognition by agents with Level 1 and Level 2 experiences are consistently higher than those performed by agents with only Level 1 experiences. The rates of correct recognition by agents with Level 1, Level 2, and Level 3 experiences are also consistently higher than those with only Level 1 and Level 2 experiences. The p-values from the two-tailed Student's t-tests on the rates of correct recognitions using different levels of experience relaxation are shown in Table 5.1. The null hypotheses are that the rates of correct recognitions are equal. Given a significance level $\alpha = 0.05$ (since all of the p-values are less than 0.05), these null hypotheses are rejected, leading to the conclusion that the differences are statistically significant. Experience relaxation does improve the rate of correct recognitions, regardless of how many experiences are in the experience base.

In order to investigate why experience relaxation helps improve situation recognition, we examined other aspects of the results. Figure 5.9 shows the rates of incorrect recognitions, and Figure 5.10 shows the rates of failing to recognize (i.e.,

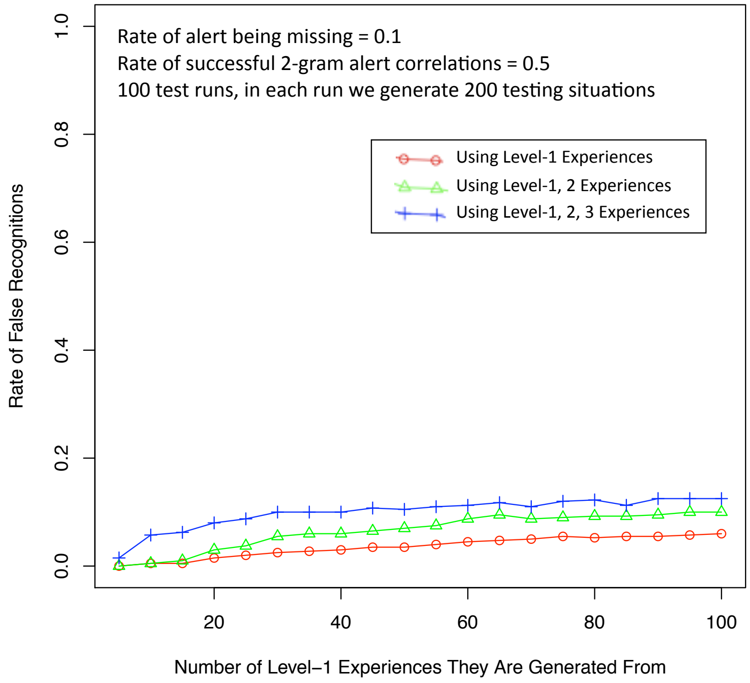


Figure 5.9. Rate of incorrect situation recognitions.

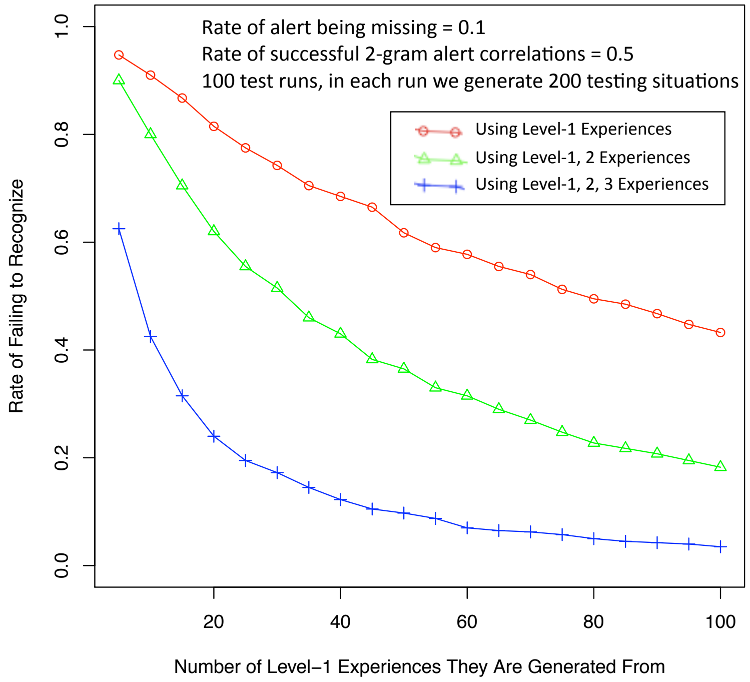


Figure 5.10. Rate of failing to recognize.

Table 5.2. p-values from the Student's t-tests on the rates of incorrect recognitions using different levels of experience relaxation.

		p-values	
		Between L1 experiences and L1+L2 experiences	Between L1+L2 experiences and L1+L2+L3 experiences
Number of level-1 experiences they are generated from	5	0.010052	0.005086
	10	0.003878	0.000311
	15	0.001583	0.000426
	20	0.000455	0.000691
	25	0.000317	0.000705
	30	0.000129	0.000765
	35	3.83E-05	0.000896
	40	6.67E-05	0.003217
	45	4.23E-05	0.003847
	50	3.52E-05	0.006089
	55	5.76E-06	0.003367
	60	1.94E-06	0.002960
	65	5.24E-06	0.006964
	70	5.10E-06	0.005482
	75	3.79E-06	0.009385
	80	4.90E-06	0.013975
	85	3.41E-06	0.014744
	90	2.75E-06	0.015904
	95	1.00E-06	0.013065
	100	1.39E-06	0.018779

Table 5.3. p-values from the Student's t-tests on the rates of failing to recognize using different levels of experience relaxation.

		p-values	
		Between L1 experiences and L1+L2 experiences	Between L1+L2 experiences and L1+L2+L3 experiences
Number of level-1 experiences they are generated from	5	1.84E-06	2.85E-15
	10	3.28E-10	2.26E-30
	15	5.80E-16	1.12E-40
	20	6.25E-21	2.72E-42
	25	4.05E-23	1.54E-41
	30	1.29E-26	5.57E-42
	35	3.16E-30	7.24E-43
	40	5.62E-29	1.01E-39
	45	1.12E-31	1.57E-39
	50	2.08E-34	6.48E-39
	55	5.21E-37	9.93E-39
	60	1.09E-40	3.10E-40
	65	1.15E-40	1.75E-38
	70	1.15E-42	4.95E-39
	75	1.15E-42	3.37E-38
	80	1.64E-44	5.80E-38
	85	2.81E-48	1.21E-38
	90	1.23E-47	3.86E-37
	95	1.04E-47	5.85E-36
	100	1.50E-48	3.47E-36

the rates suggesting that no recommendation can be generated by the agent). The p-values from the two-tailed Student's t-tests with similar null hypotheses are shown in Table 5.2 and Table 5.3. Given a significance level $\alpha = 0.05$, these null hypotheses are rejected, indicating there are significant differences between them. According to Figure 5.10, it is clear that when the number of experiences was small, a large percentage of the testing situations were not recognized by the agent. This explains the performance gaps mentioned in the previous paragraphs. The differences in the rates of failing to recognize provide an insight into why the recognition by only Level 1 experiences did not perform well compared to those experiments using all the Level 1, Level 2, and Level 3 experiences.

Ignoring the cases of failing to recognize and calculating the rate of correct recognition only from successful tests, we generated the results shown in Figure 5.11 and integrated in Figure 5.12. The average rates of correct recognition were maintained at a reasonable level regardless of the size of the experience base, although the deviations were higher when the size of experience base was small. Table 5.4 shows the p-values from the Student's t-tests to indicate whether there were significant differences in the rates of successful tests among experiments using different levels of experience relaxation; the results show that there were no significant differences. Consequently, it can be concluded that experience relaxation does not directly improve the rate of correct recognition among situations that can be processed by the agent, but it can significantly help reduce the rate of unrecognizable situations (rate of failing to recognize) due to the increased coverage.

It is also important to observe how the size of the experience base affects the

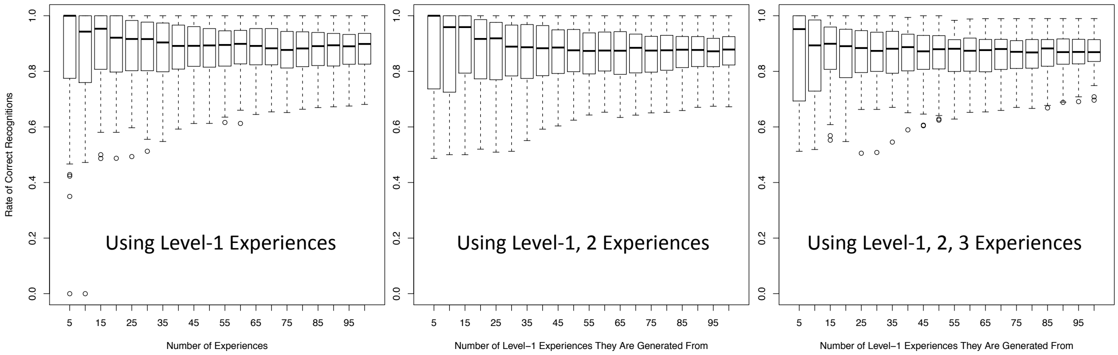


Figure 5.11. Rate of correct recognitions among successful tests.

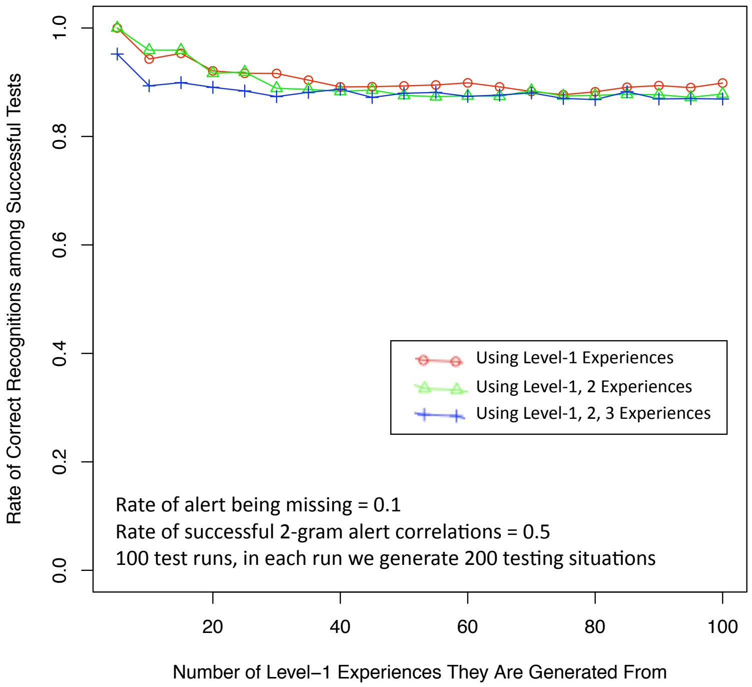
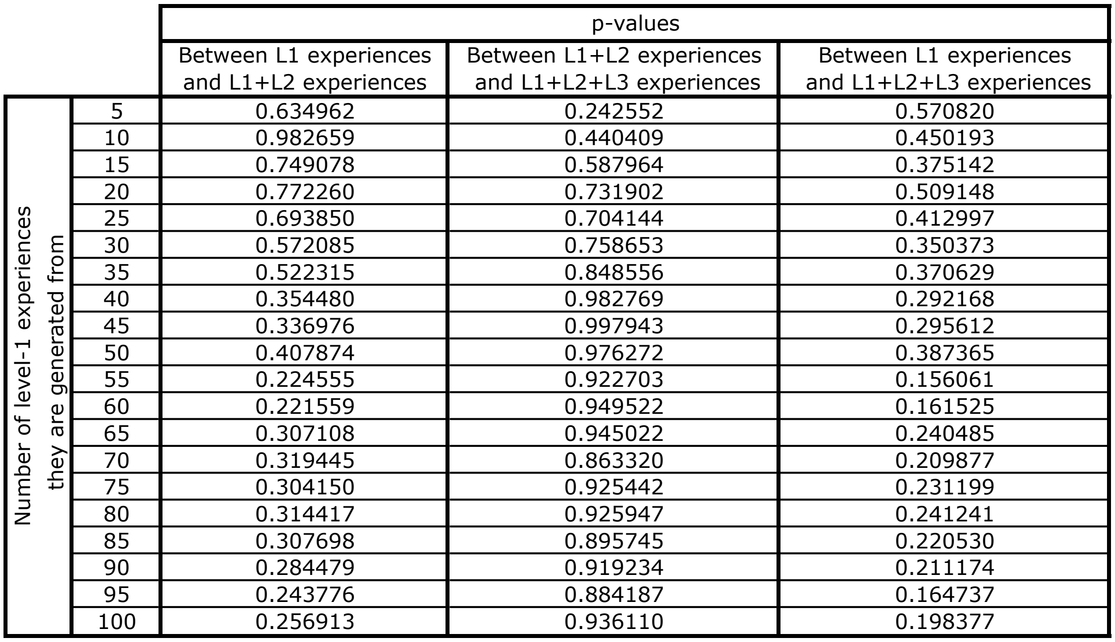


Figure 5.12. Rate of correct recognitions among successful tests.

Table 5.4. p-values from the Student's t-tests on the rates of correct recognitions among successful tests using different levels of experience relaxation.



		p-values		
		Between L1 experiences and L1+L2 experiences	Between L1+L2 experiences and L1+L2+L3 experiences	Between L1 experiences and L1+L2+L3 experiences
Number of level-1 experiences they are generated from	5	0.634962	0.242552	0.570820
	10	0.982659	0.440409	0.450193
	15	0.749078	0.587964	0.375142
	20	0.772260	0.731902	0.509148
	25	0.693850	0.704144	0.412997
	30	0.572085	0.758653	0.350373
	35	0.522315	0.848556	0.370629
	40	0.354480	0.982769	0.292168
	45	0.336976	0.997943	0.295612
	50	0.407874	0.976272	0.387365
	55	0.224555	0.922703	0.156061
	60	0.221559	0.949522	0.161525
	65	0.307108	0.945022	0.240485
	70	0.319445	0.863320	0.209877
	75	0.304150	0.925442	0.231199
	80	0.314417	0.925947	0.241241
	85	0.307698	0.895745	0.220530
	90	0.284479	0.919234	0.211174
	95	0.243776	0.884187	0.164737
	100	0.256913	0.936110	0.198377

rate of correct recognition. According to Figure 5.7 (which shows the rate of correct recognition using Level 1, Level 2, and Level 3 experiences), the rate of correct recognition increases as the size of the experience base increases. With an agent containing 100 sets of Level 1 experiences with their relaxed versions, an average of 83.50% of the situations were correctly recognized.

Additionally, it is desirable to investigate what would be a sufficiently sized experience base for making reasonably good recognitions. The p-values we obtained by performing Student's t-tests on the rates of correct recognitions given different sizes of the experience base are outlined in Table 5.5. From this table, we can see that there are significant differences when the size of experience base is less than 35, although the difference between those of size 25 and 30 is slight. When the size of the experience base is 35, the rate of correct recognition is 72.75%.

In sum, according to these experimental results, recognition using experiences with relaxation can contribute to better rates of correct recognition. This evidence and these observations confirm the usefulness of our approach in experience-based situation recognition.

Table 5.5. p-values from the Student's t-tests on the rates of correct recognitions given different numbers of Level 1 experiences they are generated from.

	Between		p-values
Number of level-1 experiences they are generated from	5	10	2.75E-15
	10	15	4.71E-12
	15	20	9.66E-06
	20	25	0.00043
	25	30	0.09634
	30	35	0.03224
	35	40	0.25042
	40	45	0.16264
	45	50	0.26501
	50	55	0.61283
	55	60	0.07226
	60	65	0.70622
	65	70	0.61003
	70	75	0.86766
	75	80	0.48836
	80	85	0.35934
	85	90	0.58541
	90	95	0.89030
	95	100	0.27865

5.2 Case Studies

In order to demonstrate the strengths and limitations of the experience-based approach, six cases based on the same scenario with different situations and emphases are shown in this section.

5.2.1 Case 1 – A Perfect Run-Time Sequence

In the first case, all alerts were generated and delivered to the analyst in the same order as the triggering events. No alerts were missing, and no false positive alerts were observed. The ground truth and the alert sequence were given as follows:

- The ground truth: $E_1 \rightarrow E_2 \rightarrow E_4 \rightarrow E_5 \rightarrow E_6$
- Alert sequence: $A_1(triggered\ by\ E_1) \rightarrow A_2(by\ E_1) \rightarrow A_3(by\ E_2) \rightarrow A_5(by\ E_4) \rightarrow A_6(by\ E_5) \rightarrow A_8(by\ E_6)$

The relationship among these observed alerts and their hidden events are outlined in Figure 5.13. This sequence can be easily matched by patterns created from experiences with no missing alerts or those relaxed from these experiences.

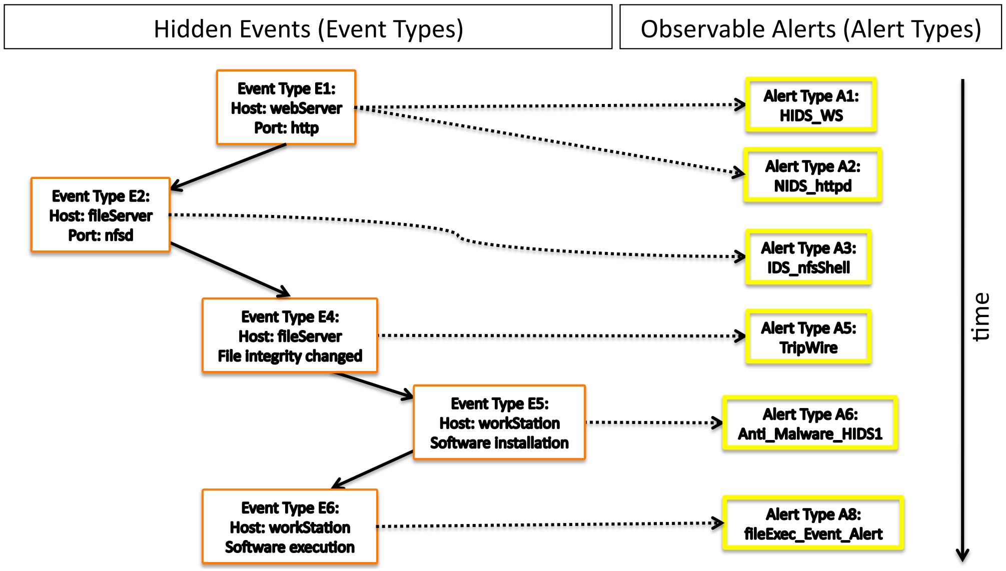

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E2: Host: fileServer Port: nfsd
Event Type E4: Host: fileServer File integrity changed
Event Type E5: Host: workStation Software installation
Event Type E6: Host: workStation Software execution
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A3: IDS_nfsShell
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A8: fileExec_Event_Alert
time

Figure 5.13. Case 1: a perfect run-time sequence.

5.2.2 Case 2 – A Sequence with a False Negative Alert

The second case (Figure 5.14) was similar to Case 1 except that it included one missing alert.

- The ground truth: $E_1 \rightarrow E_2 \rightarrow E_4 \rightarrow E_5 \rightarrow E_6$
- Alert sequence: $A_1(triggered\ by\ E_1) \rightarrow A_3(by\ E_2) \rightarrow A_5(by\ E_4) \rightarrow A_6(by\ E_5) \rightarrow A_8(by\ E_6)$

Although alert A_2 was missing, the existence of E_1 was already supported by A_1 and hence the impact was not significant. This sequence could still be detected by experiences containing A_1 in their alert patterns. It could also be matched by experiences relaxed from those containing both A_1 and A_2.

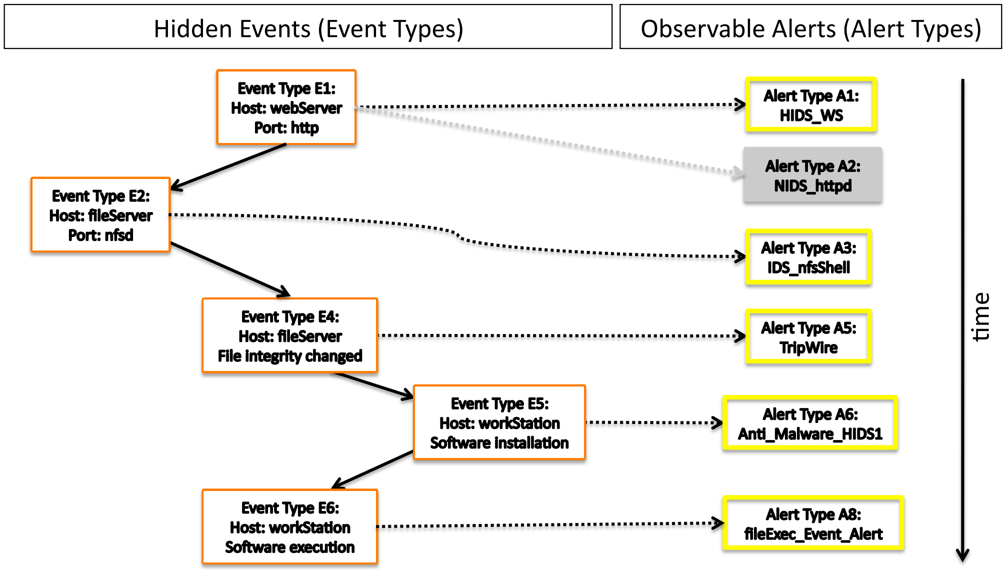

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Event Type E2: Host: fileServer Port: nfsd
Alert Type A3: IDS_nfsShell
Event Type E4: Host: fileServer File integrity changed
Alert Type A5: TripWire
Event Type E5: Host: workStation Software installation
Alert Type A6: Anti_Malware_HIDS1
Event Type E6: Host: workStation Software execution
Alert Type A8: fileExec_Event_Alert
time

Figure 5.14. Case 2: a sequence with a false negative alert.

5.2.3 Case 3 – A Sequence with another False Negative Alert

The third case (Figure 5.15) contained a missing alert, but it was the only alert of its triggering event. In addition, there was another event type which depended on the same precondition and had the same postcondition. Therefore, it became

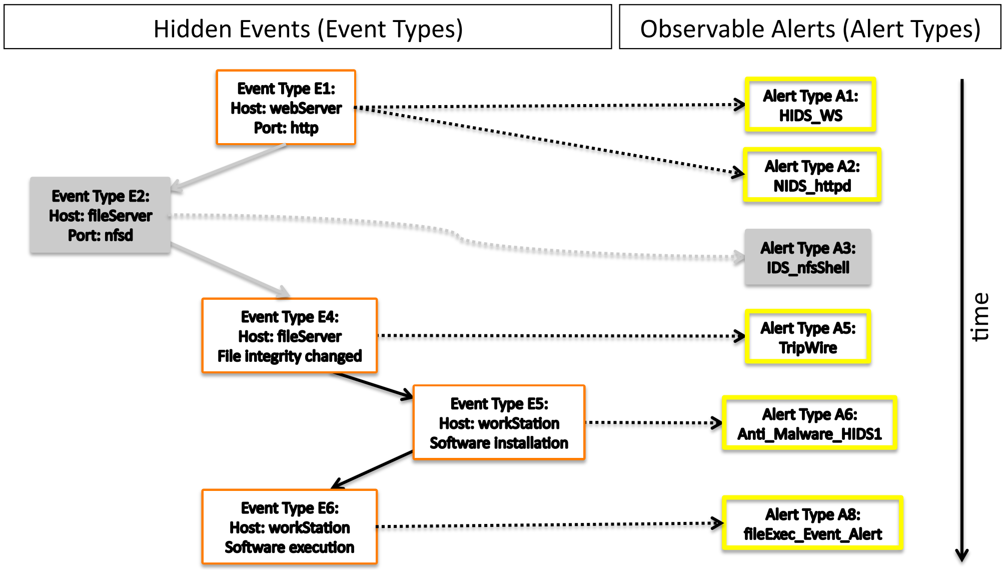

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E2: Host: fileServer Port: nfsd
Event Type E4: Host: fileServer File integrity changed
Event Type E5: Host: workStation Software installation
Event Type E6: Host: workStation Software execution
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A3: IDS_nfsShell
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A8: fileExec_Event_Alert
time

Figure 5.15. Case 3: a sequence with another false negative alert.

difficult to decide whether this attack was through the triggering event or the other event.

- The ground truth: $E_1 \rightarrow E_2 \rightarrow E_4 \rightarrow E_5 \rightarrow E_6$
- Alert sequence: $A_1(\mathit{triggered\,by}\ E_1) \rightarrow A_2(by\ E_1) \rightarrow A_5(by\ E_4) \rightarrow A_6(by\ E_5) \rightarrow A_8(by\ E_6)$

In this case, the experience base may have contained the appropriate experience to isolate and identify the situation; however, based solely on the available information, it was very challenging to distinguish which type of attack was happening due to the false negative alert.

5.2.4 Case 4 – A Sequence with a Delayed Alert

The fourth case (Figure 5.16) differed from the third in that the missing alert eventually appeared with a significant delay.

- The ground truth: $E_1 \rightarrow E_2 \rightarrow E_4 \rightarrow E_5 \rightarrow E_6$
- Alert sequence: $A_1(\mathit{triggered\,by}\ E_1) \rightarrow A_2(by\ E_1) \rightarrow A_5(by\ E_4) \rightarrow A_6(by\ E_5) \rightarrow A_3(by\ E_2) \rightarrow A_8(by\ E_6)$

When applying the experience-based recognition approach, the results may have been unreliable before this delayed alert appeared, as was the situation in Case 3. After this alert was delivered, the Recognizer changed its recommendation to report a correct result, since the information provided by the delayed alert disambiguated the entire situation.

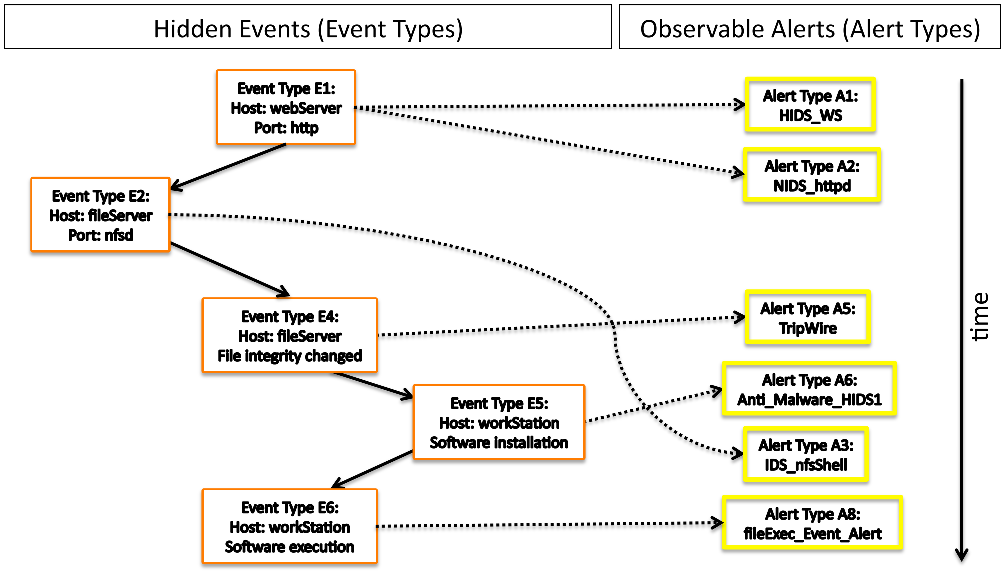

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E2: Host: fileServer Port: nfsd
Event Type E4: Host: fileServer File integrity changed
Event Type E5: Host: workStation Software installation
Event Type E6: Host: workStation Software execution
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A3: IDS_nfsShell
Alert Type A8: fileExec_Event_Alert
time

Figure 5.16. Case 4: a sequence with a delayed alert.

5.2.5 Case 5 – A Sequence with a False Positive Alert

The fifth case would confuse the analyst.

- The ground truth: $E_1 \rightarrow E_3 \rightarrow E_4 \rightarrow E_5 \rightarrow E_6$
- Alert sequence: $A_1(triggered\ by\ E_1) \rightarrow A_2(by\ E_1) \rightarrow A_4(by\ E_3) \rightarrow A_3(by\ E_2) \rightarrow A_5(by\ E_4) \rightarrow A_6(by\ E_5) \rightarrow A_8(by\ E_6)$

As shown in Figure 5.17, the existence of alert A_3 implied that there was an event E_2, but it was actually a false positive. In this case, it was very difficult to identify whether the attack was through E_2 or E_3. Experience of either attack type may have been retrieved.

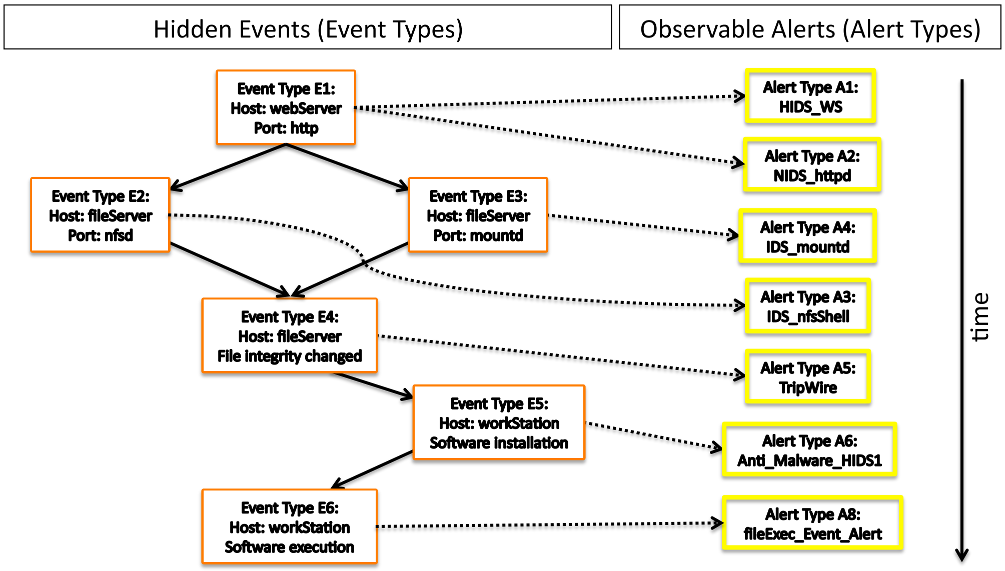

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E2: Host: fileServer Port: nfsd
Event Type E3: Host: fileServer Port: mountd
Event Type E4: Host: fileServer File integrity changed
Event Type E5: Host: workStation Software installation
Event Type E6: Host: workStation Software execution
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A4: IDS_mountd
Alert Type A3: IDS_nfsShell
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A8: fileExec_Event_Alert
time

Figure 5.17. Case 5: a sequence with a false positive alert.

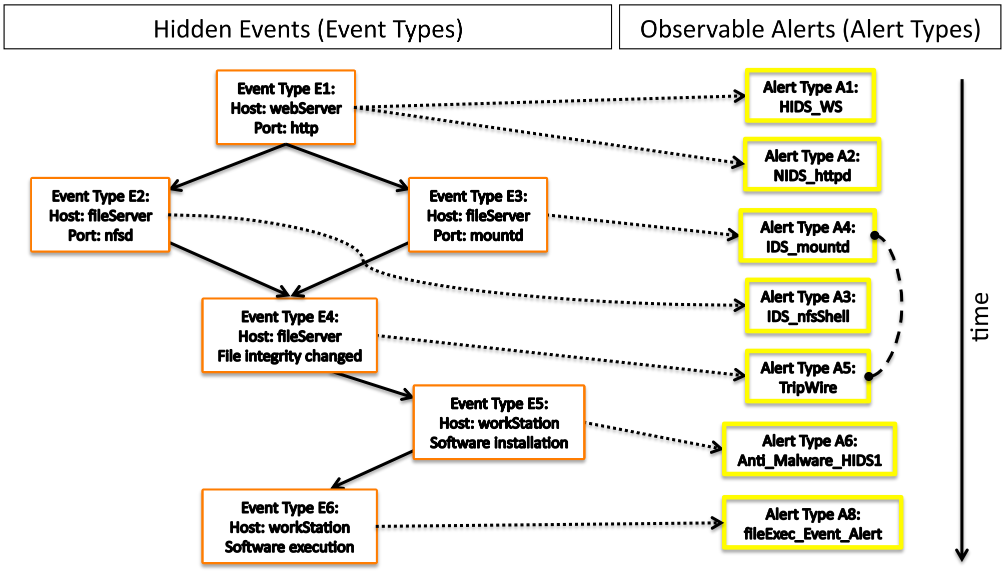

Hidden Events (Event Types)
Observable Alerts (Alert Types)
Event Type E1: Host: webServer Port: http
Event Type E2: Host: fileServer Port: nfsd
Event Type E3: Host: fileServer Port: mountd
Event Type E4: Host: fileServer File integrity changed
Event Type E5: Host: workStation Software installation
Event Type E6: Host: workStation Software execution
Alert Type A1: HIDS_WS
Alert Type A2: NIDS_httpd
Alert Type A4: IDS_mountd
Alert Type A3: IDS_nfsShell
Alert Type A5: TripWire
Alert Type A6: Anti_Malware_HIDS1
Alert Type A8: fileExec_Event_Alert
time

Figure 5.18. Case 6: a sequence with a false positive alert and correlation information.

5.2.6 Case 6 – A Sequence with a False Positive Alert and Correlation Information

Case 6 contained a false positive alert which could introduce confusion, but there was also an alert correlation report available.

- The ground truth: $E_1 \rightarrow E_3 \rightarrow E_4 \rightarrow E_5 \rightarrow E_6$
- Alert sequence: $A_1(triggered\,by\ E_1) \rightarrow A_2(by\ E_1) \rightarrow A_4(by\ E_3) \rightarrow A_3(by\ E_2)$ $\rightarrow A_5(by\ E_4) \rightarrow A_6(by\ E_5) \rightarrow A_8(by\ E_6)$; Correlated: (A_4, A_5)

This case (Figure 5.18) had additional correlation information regarding A_4 and A_5, such that the analyst could infer the intrusion was through E_3 to E_4. The confusion caused by this false positive could be resolved by alert correlation. If a similar situation had occurred before, experiences with sufficient correlation information would have helped generate correct recognition.

5.3 Remarks

From these simulations and case studies, we can see the experience-based approach provides promising results. The greater the number of situations captured by the system, the better the rate of correct recognition that can be achieved. Experience can be incrementally expanded to cover more and more cases. This coverage can be further improved if experience relaxation is applied.

As for the limitations of the experience-based approach, accurate cyber attack recognition is still a challenging issue when dealing with false negative alerts under certain confusing situations; but for false positive alerts, if alert correlation is used to help identify the relationships among different alerts, the system can provide a certain degree of correct results.

Chapter 6

Conclusions and Future Work

6.1 Main Contributions

In this dissertation, we addressed the problem of cyber situation recognition by leveraging the experience and reflection of cyber security analysts. An experience-based recognition mechanism and the idea of experience relaxation were proposed and implemented, and the experimental results demonstrated the validity of our approaches.

Going back to the research questions outlined in 1.2, our answers are listed below.

- How can we reduce the cognitive load of a cyber security analyst?
  Answer: For reducing the cognitive load of cyber security analysts, we proposed a decision support system to cover their "blind spots."

- How can we formalize the structure of a network system from the viewpoint of cyber intrusion detection?
  Answer: For the formalization of network systems, the partially observable event-alert model was proposed to capture important features from the perspective of cyber intrusion detection.

- How can we capture and leverage experts' experience to make decisions?
  Answer: To capture and leverage experts' experience to make decisions, we developed the experienced-based approach with experience relaxation.

In sum, the main contributions of this work can be summarized as follows:

- proposed an approach to enable systematic capture of experience and reflection of cyber security analysts;
- enhanced the recognition of cyber situations using the captured experiences of cyber security analysts;
- provided a knowledge-based strategy for relaxing the constraints of Horn logic-based experience patterns to enhance their utilization;
- demonstrated the benefit of experience-based cyber situation recognition through simulations.

6.2 Potentially Applicable Domains

In this dissertation, the ideas of experience-based situation recognition and experience relaxation were developed with the intent to aim at the problem domain of cyber security. These ideas can be adapted to solve other compatible problems if they share certain common features. Further research on other applicable domains can also be explored. Therefore, not only the problem of cyber security but also other applicable domains would benefit from this research.

For instance, situation awareness, not only in the virtual world but also in the real world, is an important problem [138]. Taking it in regard to military scenarios as an example, a battlefield usually involves a huge amount of dynamic information, and the participants need to make decisions based on it. Due to the information volume and other possibly unobservable but relevant events, commanders or soldiers are also subjected to their blind spots or cognitive load limitations. It would be very helpful if critical information could be consolidated through automatic mechanisms, and hidden events could be diagnosed using fast and sophisticated models. In such a dynamically changing environment, a decision support system could provide significant help and potentially save lives.

6.3 Future Work

We hope our work will be beneficial to the research community and people who are interested in related fields. To conclude the dissertation, we highlight a list of future work for the next stage of this research.

- The simulated experiments were based on identified intrusion sequences; the next step is to extend simulations with broader varieties and experiences.
- Develop further interactive learning capabilities to improve the elicitation and capture of experts' experiences and reflections.
- Extend experience-based analytics with hypothesis-based reasoning, such as doubts about the recognition and hypotheses on missing alerts.
- Investigate collaborative multi-agent systems for supporting a team of analysts.

Bibliography

[1] Tadda, G. P. and J. S. Salerno (2010) "Overview of cyber situation awareness," in *Cyber Situational Awareness* (S. Jajodia, P. Liu, V. Swarup, and C. Wang, eds.), vol. 46 of *Advances in Information Security*, chap. 2, Springer US, pp. 15–35.

[2] Kemmerer, R. A. and G. Vigna (2002) "Intrusion detection: a brief history and overview," *Computer*, **35**(4), pp. 27–30.

[3] Lunt, T. F. (1993) "A survey of intrusion detection techniques," *Computers and Security*, **12**(4), pp. 405–418.

[4] Jajodia, S. (2007) "Topological analysis of network attack vulnerability," in *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, ACM, pp. 2–2.

[5] Li, W., R. B. Vaughn, and Y. S. Dandass (2006) "An approach to model network exploitations using exploitation graphs," *Simulation*, **82**(8), pp. 523–541.

[6] Ingols, K., R. Lippmann, and K. Piwowarski (2006) "Practical attack graph generation for network defense," in *Proceedings of the 22nd Annual Computer Security Applications Conference*, pp. 121–130.

[7] Fan, X., P.-C. Chen, and J. Yen (2010) "Learning HMM-based cognitive load models for supporting human-agent teamwork," *Cognitive Systems Research*, **11**(1), pp. 108–119, Brain Informatics.

[8] ——— (2007) "Learning cognitive load models for developing team shared mental models," in *Proceedings of the 8th International Conference on Cognitive Modeling*, pp. 145–150.

[9] Agre, P. E. (1997) *Computation and Human Experience*, Cambridge University Press, New York, NY, USA.

[10] Gonzalez, C., J. F. Lerch, and C. Lebiere (2003) "Instance-based learning in dynamic decision making," *Cognitive Science*, **27**(4), pp. 591–635.

[11] Anderson, J. R. and C. Lebiere (1998) *The Atomic Components of Thought*, 1st ed., Lawrence Erlbaum.

[12] Fan, X., B. Sun, S. Sun, M. McNeese, and J. Yen (2006) "RPD-enabled agents teaming with humans for multi-context decision making," in *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 34–41.

[13] Fan, X., S. Sun, M. McNeese, and J. Yen (2005) "Extending recognition-primed decision model for human-agent collaboration," in *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 945–952.

[14] Klein, G. A. (1989) "Recognition-primed decisions," in *Advances in Man-Machine Systems Research* (W. B. Rouse, ed.), vol. 5, JAI Press, pp. 47–92.

[15] ——— (1998) "The recognition-primed decision model," in *Sources of Power: How People Make Decisions*, chap. 3, The MIT Press, pp. 15–30.

[16] Fan, X. and M. Su (2010) "Using geometric diffusions for recognition-primed multi-agent decision making," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 275–282.

[17] Yen, J., M. McNeese, T. Mullen, D. Hall, X. Fan, and P. Liu (2010) "RPD-based hypothesis reasoning for cyber situation awareness," in *Cyber Situational Awareness* (S. Jajodia, P. Liu, V. Swarup, and C. Wang, eds.), vol. 46 of *Advances in Information Security*, chap. 3, Springer US, pp. 39–49.

[18] Jha, S., O. Sheyner, and J. Wing (2002) "Two formal analyses of attack graphs," in *Proceedings of the 15th Computer Security Foundation Workshop*, pp. 49–63.

[19] Kim, G. H. and E. H. Spafford (1993) "The design and implementation of Tripwire: a file system integrity checker," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, ACM Press, pp. 18–29.

[20] Ammann, P., D. Wijesekera, and S. Kaushik (2002) "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ACM Press, pp. 217–224.

[21] Sadoddin, R. and A. Ghorbani (2006) "Alert correlation survey: framework and techniques," in *Proceedings of the 2006 International Conference on Privacy, Security and Trust*, ACM, pp. 37:1–37:10.

[22] Hay, A., D. Cid, and R. Bray (2008) *OSSEC Host-Based Intrusion Detection Guide*, Syngress Publishing.

[23] Roesch, M. (1999) "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, USENIX Association, pp. 229–238.

[24] Stakhanova, N. and A. A. Ghorbani (2010) "Managing intrusion detection rule sets," in *Proceedings of the 3rd European Workshop on System Security*, ACM, pp. 29–35.

[25] "What is the difference: viruses, worms, trojans, and bots?" `http://www.cisco.com/web/about/security/intelligence/virus-worm-diffs.html`.

[26] Pouget, F. and M. Dacier (2003) *Alert correlation: review of the state of the art*, *Tech. Rep. EURECOM+1271*, Institut Eurécom, France.

[27] Debar, H., D. Curry, and B. Feinstein (2007), "The intrusion detection message exchange format (IDMEF)," RFC 4765 (Experimental), `http://www.ietf.org/rfc/rfc4765.txt`.

[28] Feinstein, B. and G. Matthews (2007), "The intrusion detection exchange protocol (IDXP)," RFC 4767 (Experimental), `http://www.ietf.org/rfc/rfc4767.txt`.

[29] Buchheim, T., M. Erlinger, B. Feinstein, G. Matthews, R. Pollock, J. Betser, and A. Walther (2001) "Implementing the intrusion detection exchange protocol," in *Proceedings of the 17th Annual Computer Security Applications Conference*, IEEE Computer Society, pp. 32–41.

[30] "Common Vulnerability Exposure," `http://cve.mitre.org/`.

[31] "BugTraq Mailing List," `http://www.securityfocus.com/archive/1`.

[32] Cuppens, F. (2001) "Managing alerts in a multi-intrusion detection environment," in *Proceedings of the 17th Annual Computer Security Applications Conference*, IEEE Computer Society, pp. 22–31.

[33] Valdes, A. and K. Skinner (2001) "Probabilistic alert correlation," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, Springer-Verlag, pp. 54–68.

[34] Julisch, K. (2003) "Clustering intrusion detection alarms to support root cause analysis," *ACM Transactions on Information and System Security*, **6**(4), pp. 443–471.

[35] Dain, O. and R. K. Cunningham (2001) "Fusing a heterogeneous alert stream into scenarios," in *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, pp. 1–13.

[36] Zhu, B. and A. A. Ghorbani (2006) "Alert correlation for extracting attack strategies," *International Journal of Network Security*, **3**(3), pp. 244–258.

[37] Debar, H. and A. Wespi (2001) "Aggregation and correlation of intrusion-detection alerts," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, Springer-Verlag, pp. 85–103.

[38] Morin, B. and H. Debar (2003) "Correlation of intrusion symptoms: an application of chronicles," in *Proceedings of the 6th International Conference on Recent Advances in Intrusion Detection*, pp. 94–112.

[39] Eckmann, S. T., G. Vigna, and R. A. Kemmerer (2002) "STATL: an attack language for state-based intrusion detection," *Journal of Computer Security*, **10**(1-2), pp. 71–103.

[40] Cuppens, F. and R. Ortalo (2000) "LAMBDA: a language to model a database for detection of attacks," in *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection*, Springer-Verlag, pp. 197–216.

[41] Ilgun, K., R. Kemmerer, and P. Porras (1995) "State transition analysis: a rule-based intrusion detection approach," *IEEE Transactions on Sortware Engineering*, **21**(3), pp. 181–199.

[42] Lee, S.-H., H.-H. Lee, and B.-N. Noh (2004) "A rule-based intrusion alert correlation system for integrated security management," in *Computational Science - ICCS 2004* (M. Bubak, G. D. v. Albada, P. M. A. Sloot, and J. J. Dongarra, eds.), vol. 3036 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 365–372.

[43] Cheung, S., U. Lindqvist, and M. W. Fong (2003) "Modeling multistep cyber attacks for scenario recognition," in *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*, vol. 1, pp. 284–292.

[44] Cuppens, F. and A. Miège (2002) "Alert correlation in a cooperative intrusion detection framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 202–215.

[45] Ning, P., Y. Cui, and D. S. Reeves (2002) "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 245–254.

[46] Templeton, S. J. and K. Levitt (2000) "A requires/provides model for computer attacks," in *Proceedings of the 2000 Workshop on New Security Paradigms*, ACM, pp. 31–38.

[47] Qin, X. (2005) *A probabilistic-based framework for infosec alert correlation*, Ph.D. thesis, Atlanta, GA, USA.

[48] Frigault, M. and L. Wang (2008) "Measuring network security using Bayesian network-based attack graphs," in *Proceedings of the 3rd IEEE International Workshop on Security, Trust, and Privacy for Software Applications*, pp. 698–703.

[49] Ou, X., W. F. Boyer, and M. A. McQueen (2006) "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ACM, pp. 336–345.

[50] O'Hare, S., S. Noel, and K. Prole (2008) "A graph-theoretic visualization approach to network risk analysis," in *Proceedings of the 5th International Workshop on Visualization for Computer Security*, Springer-Verlag, pp. 60–67.

[51] Xie, P., J. H. Li, X. Ou, P. Liu, and R. Levy (2010) "Using Bayesian networks for cyber security analysis," in *Proceedings of the International Conference on Dependable Systems and Networks*, IEEE Computer Society, pp. 211–220.

[52] Reason, J. (1990) *Human Error*, 1st ed., Cambridge University Press.

[53] Jeffrey, R. C. (1983) *The Logic of Decision*, University of Chicago Press.

[54] Luce, R. D. and H. Raifa (1957) *Games and Decisions.*

[55] Bazerman, M. H. (1998) *Judgment in Managerial Decision Making*, John Wiley & Sons.

[56] Smith, J. E. and D. von Winterfeldt (2004) "Decision analysis in Management Science," *Management Science*, **50**(5), pp. 561–574.

[57] Kahneman, D. and A. Tversky (2000) *Choice, Values, Frames*, The Cambridge University Press.

[58] Postmes, T., R. Spears, and S. Cihangir (2001) “Quality of decision making and group norms,” *Journal of Personality and Social Psychology*, **80**(6), pp. 918–930.

[59] Gruenfeld, D. H., E. A. Mannix, K. Y. Williams, and M. A. Neale (1996) “Group composition and decision making: how member familiarity and information distribution affect process and performance,” *Organizational Behavior and Human Decision Processes*, **67**(1), pp. 1–15.

[60] Kamau, C. and D. Harorimana (2008) “Does knowledge sharing and withholding of information in organizational committees affect quality of group decision making?” in *Proceedings of the 9th European Conference on Knowledge Management*, pp. 341–348.

[61] Dean, J. W. and M. P. Sharfman (1996) “Does decision process matter? A study of strategic decision-making effectiveness,” *Academy of Management Journal*, **39**(2), pp. 368–396.

[62] Warwick, W., S. McIlwaine, R. Hutton, and P. McDermott (2001) “Developing computational models of recognition-primed decision making,” in *Proceedings of the 10th Conference on Computer Generated Forces*, pp. 323–331.

[63] Jackson, P. (1998) *Introduction to Expert Systems*, Addison-Wesley Longman Publishing Co., Inc.

[64] Ullman, J. D. (1988) *Principles of Database and Knowledge-Base Systems, Vol. I*, Computer Science Press, Inc., New York, NY, USA.

[65] Cohen, P. R. and H. J. Levesque (1991) “Teamwork,” *Noûs*, **25**(4), pp. 487–512.

[66] Airy, G., P.-C. Chen, X. Fan, J. Yen, D. Hall, M. Brogan, and T. Huynh (2006) “Collaborative RPD agents assisting decision making in Active Decision Spaces,” in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pp. 769–772.

[67] White, S. A. (2004) *Introduction to BPMN*, *Tech. rep.*, IBM Corporation.

[68] Howard, R. A. and J. E. Matheson (1984) “Influence diagrams,” in *Readings on the Principles and Applications of Decision Analysis* (R. A. Howard and J. E. Matheson, eds.), vol. 2, Strategic Decisions Group, pp. 719–762.

[69] Boutilier, C. (2005) “The influence of influence diagrams on artificial intelligence,” *Decision Analysis*, **2**(4), pp. 229–231.

[70] Shachter, R. D. (1986) "Evaluating influence diagrams," *Operations Research*, **34**(6), pp. 871–882.

[71] Zhang, N. L., R. Qi, and D. Poole (1994) "A computational theory of decision networks," *International Journal of Approximate Reasoning*, **11**, pp. 83–158.

[72] Howard, R. A. and J. E. Matheson (2005) "Influence diagram retrospective," *Decision Analysis*, **2**(3), pp. 144–147.

[73] Howard, R. A., J. E. Matheson, M. W. L. Merkhofer, A. C. Miller, and D. W. North (2006) "Comment on influence diagram retrospective," *Decision Analysis*, **3**(2), pp. 117–119.

[74] Detwarasiti, A. and R. D. Shachter (2005) "Influence diagrams for team decision analysis," *Decision Analysis*, **2**(4), pp. 207–228.

[75] Pauker, S. G. and J. B. Wong (2005) "The influence of influence diagrams in medicine," *Decision Analysis*, **2**(4), pp. 238–244.

[76] Hall, D. and J. Llinas (1997) "An introduction to multisensor data fusion," *Proceedings of the IEEE*, **85**(1), pp. 6–23.

[77] Steinberg, A. N., C. L. Bowman, and F. E. White (1999) "Revisions to the JDL data fusion model," in *Sensor Fusion: Architectures, Algorithms, and Applications III* (B. V. Dasarathy, ed.), vol. 3719, SPIE, pp. 430–441.

[78] Llinas, J., C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White (2004) "Revisiting the JDL data fusion model II," in *Proceedings of the 7th International Conference on Information Fusion* (P. Svensson and J. Schubert, eds.), pp. 1218–1230.

[79] Porter, M. E. (1998) *Competitive Advantage: Creating and Sustaining Superior Performance*, 1st ed., Free Press.

[80] Sun, S. and J. Yen (2005) "Information supply chain: a unified framework for information-sharing," in *Proceedings of IEEE International Conference on Intelligence and Security Informatics*, vol. 3495, pp. 422–428.

[81] Surana, A., S. Kumara, M. Greaves, and U. N. Raghavan (2005) "Supply-chain networks: a complex adaptive systems perspective," *International Journal of Production Research*, **43**(20), pp. 4235–4266.

[82] Choi, T. Y., K. J. Dooley, and M. Rungtusanatham (2001) "Supply networks and complex adaptive systems: control versus emergence," *Journal of Operations Management*, **19**(3), pp. 351–366.

[83] Pathak, S. D., J. M. Day, A. Nair, W. J. Sawaya, and M. M. Kristal (2007) "Complexity and adaptivity in supply networks: building supply network theory using a complex adaptive systems perspective," *Decision Sciences*, **38**(4), pp. 547–580.

[84] Thadakamaila, H., U. Raghavan, S. Kumara, and R. Albert (2004) "Survivability of multiagent-based supply networks: a topological perspect," *IEEE Intelligent Systems*, **19**(5), pp. 24–31.

[85] Rice, J. B., Jr and F. Caniato (2003) "Building a secure and resilient supply network," *Supply Chain Management Review*, **7**(5), pp. 22–30.

[86] Watts, D. J. and S. H. Strogatz (1998) "Collective dynamics of 'small-world' networks," *Nature*, **393**, pp. 440–442.

[87] Barabási, A.-L. and R. Albert (1999) "Emergence of scaling in random networks," *Science*, **286**(5439), pp. 509–512.

[88] Barabási, A.-L. (2002) *Linked*, 1st ed., Basic Books.

[89] Newman, M. E. J. (2003) "The structure and function of complex networks," *SIAM Review*, **45**(2), pp. 167–256.

[90] Boccaletti, S., V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang (2006) "Complex networks: structure and dynamics," *Physics Reports*, **424**(4-5), pp. 175–308.

[91] Costa, L. d. F., F. A. Rodrigues, G. Travieso, and P. R. V. Boas (2007) "Characterization of complex networks: a survey of measurements," *Advances in Physics*, **56**(1), pp. 167–242.

[92] Albert, R., H. Jeong, and A.-L. Barabási (2000) "Error and attack tolerance of complex networks," *Nature*, **406**(6794), pp. 378–382.

[93] Holme, P., B. J. Kim, C. N. Yoon, and S. K. Han (2002) "Attack vulnerability of complex networks," *Physical Review E*, **65**(5), p. 056109.

[94] Newman, M. E. J. (2002) "Assortative mixing in networks," *Physical Review Letters*, **89**(20), p. 208701.

[95] Bollobás, B. (2001) *Random Graphs*, 2nd ed., Cambridge University Press.

[96] Callaway, D. S., J. E. Hopcroft, J. M. Kleinberg, M. E. J. Newman, and S. H. Strogatz (2001) "Are randomly grown graphs really random?" *Physical Review E*, **64**(4), p. 041902.

[97] Grubesic, T., T. Matisziw, A. Murray, and D. Snediker (2008) "Comparative approaches for assessing network vulnerability," *International Regional Science Review*, **31**(1), pp. 88–112.

[98] Callaway, D. S., M. E. J. Newman, S. H. Strogatz, and D. J. Watts (2000) "Network robustness and fragility: percolation on random graphs," *Physical Review Letters*, **85**(25), pp. 5468–5471.

[99] Kong, Z. and E. M. Yeh (2009) "Percolation processes and wireless network resilience to degree-dependent and cascading node failures," *CoRR*, **abs/0902.4447**.

[100] Gutfraind, A. (2009) "Constructing networks for cascade resilience," *ArXiv e-prints*, `0906.0786`.

[101] Carley, K. M., J. Diesner, J. Reminga, and M. Tsvetovat (2007) "Toward an interoperable dynamic network analysis toolkit," *Decision Support Systems*, **43**(4), pp. 1324–1347.

[102] Tsvetovat, M., J. Reminga, and K. M. Carley (2004) *DyNetML: interchange format for rich social network Data*, *Tech. rep.*, Carnegie Mellon University, School of Computer Science, Institute for Software Research International.

[103] Forgy, C. L. (1982) "Rete: a fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, **19**(1), pp. 17–37.

[104] Batory, D. (1994) *The LEAPS Algorithms*, *Tech. Rep. CS-TR-94-28*, University of Texas at Austin.

[105] Lamport, L. (1978) "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, **21**(7), pp. 558–565.

[106] Kleene, S. (1956) "Representation of events in nerve nets and finite automata," in *Automata Studies* (C. Shannon and J. Mccarthy, eds.), Princeton University Press, pp. 3–42.

[107] Thompson, K. (1968) "Programming techniques: regular expression search algorithm," *Communications of the ACM*, **11**(6), pp. 419–422.

[108] Brggemann-Klein, A. (1992) "Regular expressions into finite automata," in *LATIN '92* (I. Simon, ed.), vol. 583 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 87–98.

[109] Mcilroy, M. D. (2004) "Enumerating the strings of regular languages," *Journal of Functional Programming*, **14**(5), pp. 503–518.

[110] Kumar, S., S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner (2006) "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ACM, pp. 339–350.

[111] Cox, R. (2007), "Regular expression matching can be simple and fast (but is slow in Java, Perl, PHP, Python, Ruby, ...)," `http://swtch.com/~rsc/regexp/regexp1.html`.

[112] Brown, A., S. Mantha, and T. Wakayama (1993) "A logical reconstruction of constraint relaxation hierarchies in logic programming," in *Methodologies for Intelligent Systems* (J. Komorowski and Z. Ras, eds.), vol. 689 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 362–374.

[113] Wilson, M. and A. Borning (1993) "Hierarchical constraint logic programming," *Journal of Logic Programming*, **16**(3), pp. 277–318.

[114] Govindarajan, K., B. Jayaraman, and S. Mantha (1995) "Relaxation in constraint logic languages," in *Proceedings of the ACM Symposium on Principles of Programming Languages*, pp. 91–103.

[115] Sun, S. (2006) *Using cognitively inspired agents and information supply chains to anticipate and share information for decision-making teams*, Ph.D. thesis, The Pennsylvania State University.

[116] Fan, X., R. Wang, B. Sun, S. Sun, and J. Yen (2005) "Multi-agent information dependence," in *Proceedings of the 2005 IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pp. 41–46.

[117] Yen, J., X. Fan, S. Sun, T. Hanratty, and J. Dumer (2006) "Agents with shared mental models for enhancing team decision-makings," *Journal of Decision Support Systems*, **41**(3), pp. 634–653.

[118] Friedman-Hill, E. J. (2008) *Jess, the Rule Engine for the Java Platform*, Sandia National Laboratories, `http://herzberg.ca.sandia.gov/jess/`.

[119] Sirin, E., B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz (2007) "Pellet: a practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, **5**(2), pp. 51–53.

[120] "Drools," `http://www.jboss.org/drools/`.

[121] Bechhofer, S., F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein (2004), "OWL Web ontology language reference," W3C Recommendation.

[122] Horrocks, I., P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean (2004) *SWRL: a semantic Web rule language combining OWL and RuleML*, *Tech. rep.*, World Wide Web Consortium.

[123] "RuleML," `http://www.ruleml.org/`.

[124] Chen, P.-C., G. Airy, P. Mitra, and J. Yen (2010) "Extending legacy agent knowledge base systems with semantic Web compatibilities," in *Proceedings of the 12th International Conference on Enterprise Information Systems*, pp. 131–134.

[125] Patel-Schneider, P. F. and B. Parsia (2008) *OWL 2 Web ontology language: primer*, *W3C working draft*, World Wide Web Consortium, `http://www.w3.org/TR/2008/WD-owl2-primer-20080411/`.

[126] Motik, B. and B. C. Grau (2008) *OWL 2 Web ontology language: model-theoretic semantics*, *W3C working draft*, World Wide Web Consortium, `http://www.w3.org/TR/2008/WD-owl2-semantics-20080411/`.

[127] Boley, H. and M. Kifer (2008) *RIF basic logic dialect*, *W3C working draft*, World Wide Web Consortium, `http://www.w3.org/TR/2008/WD-rif-bld-20080730/`.

[128] Meditskos, G. and N. Bassiliades (2008) "A rule-based object-oriented OWL reasoner," *IEEE Transactions on Knowledge and Data Engineering*, **20**(3), pp. 397–410.

[129] O'Connor, M. J., H. Knublauch, S. W. Tu, B. N. Grosof, M. Dean, W. E. Grosso, and M. A. Musen (2005) "Supporting rule system interoperability on the semantic Web with SWRL," in *Proceedings of the 2005 International Semantic Web Conference*, pp. 974–986.

[130] Grosof, B. N., M. D. Gandhe, M. D. G, and T. W. Finin (2002) "SweetJess: translating DamlRuleML to Jess," in *Proceedings of International Workshop on Rule Markup Languages for Business Rules on the Semantic Web.*

[131] Noy, N. F., M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen (2001) "Creating semantic Web contents with Protégé-2000," *IEEE Intelligent Systems*, **2**(16), pp. 60–71.

[132] (2007), "SWRLExtensionsBuiltIns," `http://protege.cim3.net/cgi-bin/wiki.pl?SWRLExtensionsBuiltIns`.

[133] Aamodt, A. and E. Plaza (1994) "Case-based reasoning: foundational issues, methodological variations, and system approaches," *AI Communications*, **7**(1), pp. 39–59.

[134] Triantaphyllou, E. (2000) *Multi-Criteria Decision Making Methods: A Comparative Study*, 1st ed., Springer.

[135] Huhns, M. N., M. P. Singh, M. Burstein, K. Decker, E. Durfee, T. Finin, L. Gasser, H. Goradia, N. Jennings, K. Lakkaraju, H. Nakashima, V. Parunak, J. S. Rosenschein, A. Ruvinsky, G. Sukthankar, S. Swarup, K. Sycara, M. Tambe, T. Wagner, and L. Zavala (2005) "Research directions for service-oriented multiagent systems," *IEEE Internet Computing*, **9**(6), pp. 65–70.

[136] Jonquet, C., P. Dugenie, and S. Cerri (2008) "Service-based integration of grid and multi-agent systems models," in *Service-Oriented Computing: Agents, Semantics, and Engineering* (R. Kowalczyk, M. Huhns, M. Klusch, Z. Maamar, and Q. Vo, eds.), vol. 5006 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 56–68.

[137] Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol (2009) *Discrete-Event System Simulation*, 5th ed., Prentice Hall.

[138] Endsley, M. (1995) "Toward a theory of situation awareness in dynamic systems: situation awareness," *Human factors*, **37**(1), pp. 32–64.

Vita

Po-Chun Chen

Po-Chun Chen was born in Taiwan in 1978. He completed his undergraduate studies in Information Management at National Taiwan University, Taipei, Taiwan in 2000. After finishing his military service (2000-2002), he returned to NTU to begin a master's program in Information Management, focusing on semantic Web and Web services research and receiving a degree in 2004. While working on his master's degree, Chen joined a start-up company, where he worked on every aspect of the software systems that supported its business operations. Leaving the industry in 2005, he entered the Ph.D. program in the Department of Computer Science and Engineering at The Pennsylvania State University. He worked as a Research Assistant at the university's Laboratory for Intelligent Agents from 2005 to 2010, devoting himself to several research projects relevant to intelligent agents and decision support systems. In 2010, he joined CA Technologies, where he continues his career in the software industry.