Doctoral Dissertation

Making Existing Reactive Systems Anticipatory: Methodology and Case Studies

Kai Shi

Graduate School of Science and Engineering,
Saitama University

Supervisor: Professor Jingde Cheng

June 2013

Abstract

A reactive system is a system that maintains an ongoing interaction with its environment, as opposed to obtain a final result. Various reactive systems play very important roles in modern society, such as bank transfer systems, web servers, operating systems, computer networks, air/railway traffic control systems, elevator systems, and nuclear power plant control systems. Since an accident or an attack of a critical reactive system may cause financial loss and even casualties, the biggest challenge for reactive systems is not only to ensure the system functionality, but to prevent these accidents and attacks.

A traditional reactive system is usually passive, i.e., the system can only perform those operations in response to instructions explicitly issued by users or application programs, but have no ability to do something actively and anticipatorily by themselves. From the viewpoint of safety engineering and security engineering, a traditional passive reactive system only has some quite weak capability to defend accidents and attacks from its external computing environment. In order to prevent accidents/attacks beforehand, it is desired that a reactive system is anticipatory, i.e., the system can detect and predict omens of accidents/attacks anticipatorily and then take some actions to inform its users and perform some operations to defend accidents/attacks by itself. Therefore, from the viewpoints of high safety and high security, any critical reactive system should be anticipatory.

In order to build practically useful reactive systems with the ability of anticipation, Cheng proposed anticipatory reasoning-reacting system as a certain class of computing anticipatory systems, which is a computing system can predict based on the predictive model then take anticipatory actions according to the predictions as well as take reactive actions to the current situation based on the behavioral model. The most important features of proposed anticipatory reasoning-reacting systems are: (1) both the prediction and decision making are base on logic-based forward reasoning, and (2) an anticipatory reasoning-reacting system is a simple extension of a reactive system.

However, there are many critical existing systems which perform routine operations well but do not have the ability of anticipation to handle accidents and attacks. These systems no longer satisfy the requirements of high safety and high security. Furthermore, it is impractical, if not impossible, to rebuild the whole system to be anticipatory, because reimplementation of the whole of a system results in high cost. On the other hand, it is not necessary and economic to rebuild the whole of an existing reactive system with anticipatory ability, if it is possible to extend certain types of existing systems with anticipatory ability without affecting its original functions. However, by now, there is no study about which kind of reactive systems can be extend to be anticipatory, and no study about how to add anticipatory ability without affecting or with minor affecting the original system.

Therefore, we argued it is possible to extend an existing reactive system (called legacy system) to be an anticipatory reasoning-reacting system (called target system) without reimplementing the whole of the system, and proposed a general

methodology to realize such an extension. In this research, we first investigated and analyzed current reactive systems, in order to find out how the current reactive systems ensure the safety/security, and to find out which kind of existing reactive systems can be made anticipatory. Based on the analysis, we specified some requirements for the legacy systems which can be extend to be anticipatory. Then we discussed why it is possible to extend a reactive system anticipatory without affecting its original function. Based on the above work, we proposed a general architecture of anticipatory reasoning-reacting system for the extension, and a general process to extend an existing reactive system to be anticipatory. The main phases of the process include: (1) to analyze the target domain, aiming to find out possible accidents/attacks in the target, as well as their causation, formation process, and the consequence, (2) to analysis the legacy system, aiming to ensure the original function of the system, find out how to get information used for detection and prediction from the system, and find out which function of the system can be used as anticipatory actions, (3) to define requirements of the target system, (4) to construct the anticipatory models, which underline both predicting and choosing anticipatory actions, and (5) to prepare the anticipatory components, especially to design and implement the special components for the certain target domain, and to integrate all anticipatory components with the legacy system. The novelty of the methodology is that it does not affect the system's original function, and it can deal with various reactive systems by using the same process.

In order to show the effectiveness and usefulness of our methodology, we chose three typical reactive systems as case studies: emergency elevator evacuation systems, runway incursion prevention systems, and computing application servers, then applied the methodology to extend these systems to be anticipatory. For each case study, we elaborated its motivation, system design, system implementation, and evaluation, as well as showed how to apply the methodology and the advantage of the certain anticipatory reasoning-reacting system in that case study. After presented the case studies, we evaluated the methodology from viewpoint of generality and particularity.

This work has following contributions. First, we conceived a new approach to improve existing reactive systems safety and/or security by extending the system with anticipatory ability. Second, we proposed a general methodology, which can extend various reactive systems with anticipatory ability, and we showed the effectiveness of the methodology by applying the methodology to different case studies. Third, we built three practical ARRSs in the case studies, thus showed the practical usefulness of anticipation and ARRSs for safety and security. Previous studies of ARRSs are mainly theoretical, such as formal definition, architecture design, mechanism of prediction and decision-making, and prototype implementation, thus, there was a gap between those theoretical work and practical ARRSs. Whereas, in this work, we built practical ARRSs, as well as solve several practical problems when building practical ARRSs.

This thesis is organized as follows. Chapter 1 presents the background, motivation, and purpose of this work. Chapter 2 surveys reactive systems, and analyzes the feasibility to extend an existing reactive system to be anticipatory. Chapter 3 gives a review of anticipatory reasoning-reacting systems. Chapter 4 presents the

methodology to extend the existing reactive systems to be anticipatory. Chapter 5 shows a case study of emergency elevator evacuation systems. Chapter 6 shows a case study of runway incursion prevention systems. Chapter 7 shows a case study of computing application servers. Chapter 8 discusses the generality and particularity of the methodology. Concluding remarks are given in chapter 9.

Acknowledgments

I would like to gratefully and sincerely thank my thesis supervisor Professor Jingde Cheng for his guidance, understanding, and invaluable support through my graduate studies. I am also grateful to my dissertation committee: Professor Norihiko Yoshida, Associate Professor Toshinori Yamada, Associate Professor Noriaki Yoshiura, and Associate Professor Takashi Horiyama for their support, valuable feedback, and insightful ideas to this research. I am really grateful to Assistant Professor Yuichi Goto for teaching me how to do research and helping me in all respect. I would like to thank Professor Zhiliang Zhu, and Associate Professor Dancheng Li for their invaluable support through my graduate studies. I would like to thank Bo Wang, Hongbiao Gao, Da Bao, and other AISE lab members who have helped me in my doctoral research. I would also like to thank Dr. Ang Li and Dr. Bo Zhang for their help for my research. Finally, I would like to thank my wife and my daughter for their understanding and invaluable support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

A reactive system maintains an ongoing interaction with its environment, as opposed to obtain a final result [85]. Various reactive systems play very important roles in modern society, such as bank transfer systems, web servers, operating systems, computer networks, air/railway traffic control systems, elevator systems, and nuclear power plant control systems. Since an accident or an attack of a critical reactive system may cause financial loss and even casualties, the biggest challenge for reactive systems is not only to ensure the system functionality, but to prevent these accidents and attacks [17, 120].

A traditional reactive system is passive, i.e., it only performs those operations in response to instructions explicitly issued by users or application programs, but have no ability to do something actively and anticipatorily by itself. Therefore, a passive reactive system only has some quite weak capability to defend accidents and attacks from its external environment. In order to prevent accidents/attacks beforehand, it is desired that a reactive system is *anticipatory*, i.e., the system should be able to detect and predict accidents/attacks, take some actions to inform its users, and perform some operations to defend the system from possible accidents/attacks anticipatorily. From the viewpoints of high safety and high security, any critical reactive system should be anticipatory [38].

To build practical anticipatory systems [154] with high safety and high security, Cheng proposed *anticipatory reasoning-reacting system* (ARRS) [38], which is a computing system that can predict based on the predictive model, then take anticipatory actions according to the predictions as well as take reactive actions to the current situation based on the behavioral model. In other word, an ARRS is a reactive system with ability of anticipation. Some prototypes of ARRSs were implemented [105, 159].

The problem is that most of existing reactive systems are not anticipatory, furthermore, it is impractical, but not impossible, to rebuild them to be anticipatory, because reimplementation of the whole of a system results in high cost. If we can extend an existing reactive system with anticipatory ability but preserving the system's original functions, we can improve the system safety or security at a relatively small cost. However, there is no study about whether it is possible to

extend an existing reactive system to be an anticipatory and how to realize such an extension.

1.2 Purposes and objectives

Our purpose is to propose a general methodology to extend an existing legacy reactive system with anticipatory ability without reimplementing the whole of the legacy system. By extending an existing reactive system anticipatory using our methodology, we can get a new generation system, indeed an ARRS, with higher safety and higher security.

Our work involves following objectives. First, we show existing reactive systems' need for extension with anticipation and feasibility of the extension, based on the analysis of current reactive systems and their development approaches. Second, we propose and elaborate the methodology for extension. Third, to show the effectiveness and usefulness of our methodology, we apply our methodology to different reactive systems. In this work, we chose three typical reactive systems as case studies: emergency elevator evacuation systems, runway incursion prevention systems, and computing application servers. For each case study, we elaborate its motivation, system design, system implementation, and evaluation, as well as showed how to apply the methodology and the advantage of the certain anticipatory reasoning-reacting system in that case study. Finally, after presented the case studies, we evaluate the methodology from viewpoint of generality and particularity.

1.3 Structure of this thesis

This thesis is organized as follows. Chapter 1 presents the background, motivation, and purpose of this work. Chapter 2 surveys reactive systems, and analyzes the feasibility to extend an existing reactive system to be anticipatory. Chapter 3 gives a review of anticipatory reasoning-reacting systems. Chapter 4 presents the methodology to extend the existing reactive systems to be anticipatory. Chapter 5 shows a case study of emergency elevator evacuation systems. Chapter 6 shows a case study of runway incursion prevention systems. Chapter 7 shows a case study of computing application servers. Chapter 8 discusses the generality and particularity of the methodology. Concluding remarks are given in chapter 9.

Chapter 2

Reactive systems and feasibility analysis of extension

2.1 Overview

Harel and Pnueli introduced the notation of reactive systems in 1985 [85]. According to Harel and Pnueli, we can divide systems into two classes, transformational and reactive systems [85]. A transformational system accepts inputs, performs transformations on them and produces output, i.e., a final result on termination, shown in Figure 2.1 [85]. By contrast, a *reactive system* is a computing system whose role is to maintain an ongoing interaction with its environment rather than to compute some final value on termination [85, 126, 145]. Figure 2.2 shows a reactive system M is a "black cactus" which "thorns" are the interface elements comprising the set E [85]. The behaviors of M is a subset of the words over E [85].

Reactive systems have the following characteristics [178].

- A reactive system continuously interacts with its environment.
- The interactions between the system and its environment is nonterminating. The termination of the system is usually considered to be a failure.
- The system must respond to external events of the environment when they occur. To respond to external stimuli concurrently, the system can be concurrent [77, 126], as well as be sequential with the ability to respond to interrupts [145, 178].

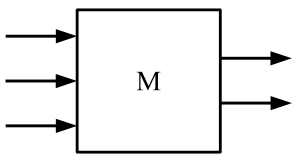

Figure 2.1: A transformational system as a black box

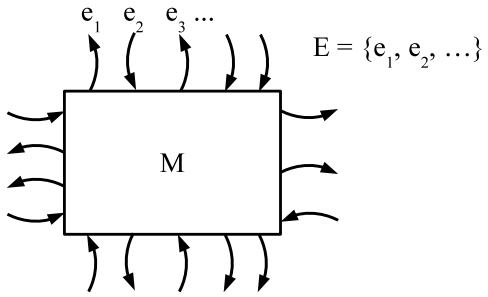

Figure 2.2: A reactive system as a "black cactus"

- The system must response to an external event under a stringent time requirements.
- The response of the system rely on the sequences of the input events. The response may leave the system in a different state than it was before. In other words, the response of the system depends on its current state and the external event that it responds to.
- The response consists of enabling, enforcing, or prohibiting communication or behavior in its environment.
- The behavior of the system often consists of a lot of interacting processes that operate in parallel.

2.2 Functions and classification

A reactive system can have three kinds of functions [178]:

- Informative: To provide information about the subject domain, such as to answer questions and produce reports.
- Directive: To direct the entities in the subject domain, such as to control or guide a physical entity.
- Manipulative: To manipulate lexical items in the subject domain, such as to create, remove, or modify lexical items.

An informative function requires system receives a message from a sender and responds to the same sender. Such a message may or may not be part of the subject domain. The message is usually a query or update request, and the response is an answer or update confirmation.

A directive function constrains or influences the subject domain. The sender of the message and the receiver of the response are usually different entities in the subject domain. Other possible sources and destinations of messages may be a system operator and a system client, who are not in the subject domain. They may send messages to the system that lead to directives sent to subject domain entities.

A manipulative function constructs and maintains a domain of lexical items. lexical items represent conceptual entities such as texts, graphs, contracts, and specifications. The source of the messages about the subject domain is some entity in the environment, such as a system user, and the response is usually an update of the constructed lexical items plus a feedback to the source and possibly to other external entities. In this case, the manipulated lexical domain entities are part of the reactive system.

Generally, one reactive system has one significant function, thus we can classify the reactive systems by their functions. Accordingly, there are three classes of reactive systems, i.e., informative reactive systems, directive reactive systems, and manipulative reactive systems.

2.3 Safety and information security

Safety and information security are typical emergent properties of reactive systems. Emergent properties [37] are properties of the system as a whole and only emerge once all of its individual sub-systems have been integrated [112]. The concept of emergence is the idea that at a given level of complexity, some properties of that level are irreducible [120].

Safety is freedom of risk [52] (accidents or losses) [120]. A safe system is one that will never do anything bad, that the definition of what is "bad" is application-dependent [52]. Safety is the most critical emergent property of typical directive reactive systems such as industrial control systems, aviation systems, medical care systems, and weapons.

Generally, *information security* is prevention of or protection against access to information by unauthorized recipients or intention but unauthorized destruction or alteration of that information [52]. Manipulative reactive systems are widely used by governments, financial institutions, hospitals, institutes, and corporations to manage lexical entities about their employees, finance, customers, research, and products. These systems and information are often critical assets that support the mission of an organization [75]. Protecting them can be as critical as protecting other organizational resources [75]. Moreover, today, information security of information systems is not only about confidentiality, integrity and availability, but also about ensuring that the systems are predictably dependable in the face of all sorts of malice [17].

2.4 Approaches to develop reactive systems

For reactive systems development, the problem rooted in the notion of the behavior of a system, i.e., it is difficulty of describing reactive behavior in way that are clear and realistic, and at the same time formal and rigorous, sufficiently so to be amenable to detailed computerized simulation [81, 85]. Consequently, most of approaches of so-called "reactive systems" focus on formal methods, i.e., formal specification, verification, and development of reactive systems, whose purpose is to ensure the correctness of the systems.

Although several important approaches have been proposed, these approaches has not yet been satisfactorily solved the problem of reactive systems development [77, 81], such as state-machines/state-diagrams (such as [56, 65, 66, 95, 136, 168, 169]), communicating finite-state machines [152], augmented transition networks [176, 180], structured analysis and design technique (SADT) [155], higher order software (HOS) [80], petri net [140], Milner's calculus of communicating systems (CCS) [129], Hoare's communicating sequential processes (CSP) [88], and Zave's sequence diagrams [183], Thus, although some people use these methods to develop reactive systems [14], we do not discuss them here.

2.4.1 Statecharts

The the visual language *Statecharts* [81] was created by Harel [82, 89]. In fact, the work on reactive systems began with the discussion of Statecharts [82, 89]. Harel suggested Statecharts is useful for particular kinds of systems that interact frequently with one another and do not do heavy computation, then Pnueli coin the terminology "reactive systems" [82, 89].

Statecharts is the first and most popular formal language for the design of reactive systems. Statecharts is not only useful for modeling system behavior in the structural analysis paradigm, but also part of a fully executable language set for modeling object-oriented systems [83]. Besides, Statecharts is part of the unified modeling language (UML) since UML 1.x [93].

Harel's Statecharts extended conventional state-transition diagrams with hierarchy, concurrency and communication, in order to provide a formalism for specifying reactive behavior [81]. In a word,

$$\begin{aligned} \textit{Statecharts} =& \textit{state-diagrams} + \textit{depth} + \textit{orthogonality} \\ &+ \textit{broadcast-communication}\,[81]. \end{aligned}$$

Based on Statecharts, a software tool called *statemate* [84] was developed. Statemate is a graphical working environment, intended for the specification, analysis, design and documentation for large and complex reactive systems [84]. It can be used to prepare, analyze and debug precise diagramatic descriptions of the systems from three points of view, capturing, structure, functionality and behavior [84]. Statemate was "the first commercial computer-aided software engineering tool to successfully overcome the challenges of complex interactive, real time computer systems, known as reactive systems (2007 ACM software system award announcement [89]). "

2.4.2 Temporal logic

Pnueli's landmark paper, "The Temporal Logic of Programs" [144], marked a crucial tuning point in the verification of concurrent programs and reactive systems [89]. By introduce the temporal logic to the field of formal methods, Pnueli gave researcher a set of tools to specify and reason about the ongoing behavior of programs [89].

The "classical" approach to the use of temporal logic for reasoning about reactive programs is a manual one. Emerson et al. proposed automated approach to reasoning about reactive systems especially for model checking and testing satisfiability [49, 57, 58].

2.4.3 Bigraphical reactive systems

Milner et al. proposed bigraphical reactive systems (BRS) [119, 128] as a unifying theory of process models and as a tool for reasoning about ubiquitous computing. BRS is useful for design engineers and analysts with the help of computerized visualization backed by rigorous machine-assisted verification [119].

A BRS involves bigraphs, in which the nesting of nodes represents locality, independently of the edges connecting them; it also allows bigraphs to reconfigure themselves [97]. BRSs aim to provide a uniform way to model spatially distributed systems that both compute and communicate [97].

2.4.4 Synchronous programming of reactive systems

Synchronous languages [25, 77] were designed to allow a program to be considered as instantaneously reacting to external events [77]. Synchronous languages were proposed to cover up two deficiency of Statecharts: (1) Statecharts are specification and design formalism, not programming language, (2) many features of synchronous model are present in Statecharts, but determinism is not ensured, and many semantic problems were raised [174]. Three synchronous programming languages was proposed [78], include Esterel [26], Signal [118], and Lustre [79]. A purely synchronous variant of Statecharts named Argos [127] was proposed. Based on Argos, a graphical version of Esterel named SyncCharts [18] was proposed.

2.4.5 Fault tree analysis

Fault tree analysis is a traditional analytic tool for the reliability and safety of complex systems. It is used for analyzing, visually displaying and evaluating failure paths in a system, thereby providing a mechanism for effective system level risk evaluations [59]. Because safety analysis activities are particularly critical in the case of reactive systems, fault tree analysis can be used to develop reactive systems [29, 150].

2.4.6 Object-oriented approaches

The main innovations of object-oriented specification of reactive systems are in raising the level of abstraction in dealing with concurrency, and in supporting the structuring and derivation of operational specifications, to enhance modifiability, scalability, and reusability [96]. Object-oriented specification [96] and object-printed model [19, 138] for reactive systems were proposed.

On the other hand, because of extensive use of object-oriented design, there are a lot of object-oriented reactive systems are developed. Both contracts [87]

and Statecharts can be used to specify behavioral compositions in object-oriented systems. There are also some test method for object-oriented reactive systems, such as Spec Explorer [35].

2.4.7 Secure reactive systems

Most practically relevant cryptographic systems are reactive [142]. Thus actually all systems secure against active attacks must be defined reactively because such an attack presupposes that the system gets several inputs or makes several outputs [142].

Pfitzmann, Waidner, et al. proposed cryptographic security of reactive systems [141], a model for asynchronous reactive systems to secure message transmission [143], and a general composition theorem for secure reactive systems [21].

2.5 Need of extension with anticipation

A traditional reactive system is passive, i.e., it only performs those operations in response to instructions explicitly issued by users or application programs, but have no ability to do something actively and anticipatorily by itself. Therefore, a passive reactive system only has some quite weak capability to defend accidents and attacks from its external environment. In order to prevent accidents/attacks beforehand, it is desired that a reactive system is *anticipatory*, i.e., the system should be able to detect and predict accidents/attacks, take some actions to inform its users, and perform some operations to defend the system from possible accidents/attacks anticipatorily. From the viewpoints of high safety and high security, any critical reactive system should be anticipatory [38].

The motivation of anticipation is to deal with safety and information security, but traditional approaches to develop reactive systems are not. As we mentioned in section 2.4, current approaches to develop reactive systems focus on the behavior of a system, i.e., the functions of a system, but not emergent properties, such as safety and information security. Moreover, these formal methods are high-cost and difficult to use, thus people developed many existing systems without using these formal methods.

There are many critical existing systems which perform routine operations well but do not have the ability of anticipation to handle accidents and attacks. These systems no longer satisfy the requirements of high safety and high security. Furthermore, it is impractical, if not impossible, to rebuild the whole system to be anticipatory, because reimplementation of the whole of a system results in high cost. On the other hand, it is not necessary and economic to rebuild the whole of an existing reactive system with anticipatory ability, if it is possible to extend certain types of existing systems with anticipatory ability without affecting its original functions.

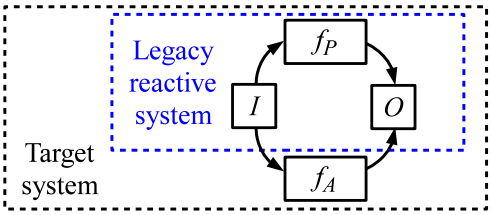

Figure 2.3: Extending a legacy reactive system with anticipatory function

2.6 Feasibility analysis of extension

Any reactive system M has a set I of inputs (e.g., inputs that represent events and conditions produced by the environment), a set O of outputs (e.g., action instructions produced by M), and a reactive function f_P mapping non-empty sequences of subset of I into a subset of O [85, 146]. Many reactive systems also have some implemented mechanisms to take action instructions to prevent an ongoing/potential accident/attack, such as applying the brake in a railway system, and blocking an intruder by configuring a firewall in computer networks. However, a traditional reactive system cannot take these action instructions anticipatorily, because f_P is passive. Therefore, we can add an additional anticipatory function f_A shown in figure 2.3, which can detect and predict accidents/attacks base on I, then choose appropriate action instructions to handle these accidents/attacks, i.e., anticipatory actions, which can be taken by the original reactive system. For most concurrent systems, the f_A only carries out some additional action instructions, while the system's original reactive function is kept, which means the original reactive actions and the anticipatory actions are taken simultaneously. For those systems that anticipatory actions may affect the system's original reactive actions such as a critical sequence of actions, in most cases, the system's original reactive actions have a higher priority thus the anticipatory actions must wait, because we do not want to affect the system's original functions. However, in some situation, the original reactive actions may make against the safety or security of the system, the original actions should be canceled, while the system's original functions are still kept in most of the time. We call the original reactive system *legacy system*, and call the extended system *target system*, as the figure 2.3 shows.

As we discussed above, if we want to extend a legacy reactive system with anticipation, the system must satisfy the following requirements.

- There must be some approaches to perceive adequate subset of I of the legacy system for detection and prediction.
- The legacy system must have some implemented mechanism to take actions to handle the ongoing/potential accidents/attacks.

Besides, for some extreme real time systems, it may be too late to predict and take actions before the accidents/attacks cause pernicious consequences, thus such systems are not suitable to extend with anticipation.

Chapter 3

Anticipatory reasoning-reacting systems

3.1 Logic-based forward reasoning on ARRS

The following content in the section is according to [70].

Anticipation is the action of taking into possession of some thing or things beforehand, or acting in advance so as preclude the action of another. It is a notion must relate to two parties such that the party taking anticipation acts in advance of a proper time earlier than the time when another party acts. To implement the facility of anticipation, we can naturally find following issues: how to predict future event or events, and how to take next actions. For the facilities, a prediction method and a decision-making method with forward reasoning based on strong relevant logic systems are proposed [38, 40, 44, 46, 106, 107].

Reasoning is the process of drawing new conclusions from given premises, which are already known facts or previously assumed hypotheses (Note that how to define the notion of 'new' formally and satisfactorily is still a difficult open problem until now). In general, a reasoning consists of a number of arguments (or inferences) in some order. An argument is a set of statements (or declarative sentences) of which one statement is intended as the conclusion, and one or more statements, called 'premises,' are intended to provide some evidence for the conclusion. An argument is a conclusion standing in relation to its supporting evidence. In an argument, a claim is being made that there is some sort of evidential relation between its premises and its conclusion: the conclusion is supposed to follow from the premises, or equivalently, the premises are supposed to entail the conclusion. Therefore, the correctness of an argument is a matter of the connection between its premises and its conclusion, and concerns the strength of the relation between them (Note that the correctness of an argument depends neither on whether the premises are really true or not, nor on whether the conclusion is really true or not). Thus, there are some fundamental questions: What is the criterion by which one can decide whether the conclusion of an argument or a reasoning really does follow from its premises or not? Is there the only one criterion, or are there many criteria? If there are many criteria, what are the intrinsic differences between them? It

is logic that deals with the validity of argument and reasoning in general. A logically valid reasoning is a reasoning such that its arguments are justified based on some logical validity criterion provided by a logic system in order to obtain correct conclusions (Note that here the term 'correct' does not necessarily mean 'true'.). Today, there are so many different logic systems motivated by various philosophical considerations. As a result, a reasoning may be valid on one logical validity criterion but invalid on another.

A *formal logic system* L is a doublet $(F(L), \vdash_L)$ where $F(L)$ is a formal language which is the set of all *well-formed formulas* of L, and $\vdash_L$ is a *logical consequence relation* of L such that for $P \subseteq F(L)$ and $c \in F(L)$, $P \vdash_L c$ means that within the framework of L, c is a valid conclusion of premises P. For a formal logic system $(F(L), \vdash_L)$, a *logical theorem* t is a formula of L such that $\emptyset \vdash_L t$ where $\emptyset$ is the empty$Th(L)$ set. We use $Th(L)$ to denote the set of all logical theorems of L. $Th(L)$ is completely determined by the logical consequence relation $\vdash_L$. According to the representation of the logical consequence relation of a logic, the logic can be represented as a Hilbert style formal system, a Gentzen natural deduction system, a Gentzen sequent calculus system, or other type of formal system. Let $(F(L), \vdash_L)$ be a formal logic system and $P \subseteq F(L)$ be a non-empty set of sentences (i.e. closed well-formed formulas).

A *formal theory* with premises P based on L, called a *L-theory with premises P* and denoted by $T_L(P)$, is defined as $T_L(P) =_{df} Th(L) \cup Th^e_L(P)$, and $Th^e_L(P) =_{df} \{\text{et} \mid P \vdash_L \text{et and et} \notin Th(L)\}$ where $Th(L)$ and $Th^e_L(P)$ are called the *logical part* and the *empirical part* of the formal theory, respectively, and any element of $Th^e_L(P)$ is called an *empirical theorem* of the formal theory.

For a formal logic system where the notion of conditional is represented by primitive connective entailment "$\Rightarrow$", a formula is called a *zero degree formula* if and only if there is no occurrence of $\Rightarrow$ in it; a formula of the form $A \Rightarrow B$ or $Q(A \Rightarrow B)$ where Q is the quantifier prefix of $A \Rightarrow B$ is called a *first degree conditional* if and only if both A and B are zero degree formula; a formula A is called a *first degree formula* if and only if it satisfies one of the following conditions: (1) A is a first degree conditional, (2) A is in the form $+B$ or $Q(+B)$ where $+$ is a one-place connective such as negation and so on and Q is the quantifier prefix of $+B$ such that B is a first degree formula, and (3) A is in the form $B * C$ or $Q(B * C)$ where $*$ is a non-conditional two-place connective such as conjunction or disjunction and so on and Q is the quantifier prefix of $B * C$ such that both of B and C is first degree formulas, or one of B and C is a first degree formula and another is a zero degree formula.

The notion of degree of nested conditional can also be generally defined as follows. Let A, B, C are formulas. The degree of conditional in A, denoted by $D_{\Rightarrow}(A)$, is inductively defined as follows: (1) $D_{\Rightarrow}(A) = 0$ if and only if there is no occurrence of $\Rightarrow$ in A, (2) if A is a conditional of the form $B \Rightarrow C$, then $D_{\Rightarrow}(A) = max\{D_{\Rightarrow}(B), D_{\Rightarrow}(C)\} + 1$, (3) if A is in the form $+B$ where $+$ is a one-place connective such as negation and so on, then $D_{\Rightarrow}(A) = D_{\Rightarrow}(B)$, (4) if A is in the form $B * C$ where $*$ is a non-conditional two-place connective such as conjunction or disjunction and so on, then $D_{\Rightarrow}(A) = max\{D_{\Rightarrow}(B), D_{\Rightarrow}(C)\}$, and (5) if A is in the form QB where Q is the quantifier prefix of B, then $D_{\Rightarrow}(A) = D_{\Rightarrow}(B)$. Let

k be a natural number. A formula A is called a k^{th} *degree formula* if and only if $D_{\Rightarrow}(A) = k$, in particular, A is called a k^{th} *degree conditional* if it is a conditional.

Let $(F(L), \vdash_L)$ be a formal logic system and k be a natural number. The k^{th} *degree fragment* of L, denoted by $Th^k(L)$, is a set of logical theorems of L that is inductively defined as follows (in the terms of Hilbert-style formal systems): (1) if A is an axiom of L and $D_{\Rightarrow}(A) \leq k$, then $A \in Th^k(L)$, (2) if A is the result of applying an inference rule of L to some members of $Th^k(L)$ and $D_{\Rightarrow}(A) \leq k$, then $A \in Th^k(L)$, and (3) Nothing else are members of $Th^k(L)$, i.e., only those obtained from repeated applications of (1) and (2) are members of $Th^k(L)$.

Let $(F(L), \vdash_L)$ be a formal logic system, $P \subset F(L)$ $(P \neq \emptyset)$, and k and j be two natural numbers. A formula A is said to be j^{th}*-degree-deducible from* P *based on* $Th^k(L)$ if and only if there is an finite sequence of formulas $f_1, \ldots, f_n$ such that $f_n = A$ and for all $i(i \leq n)$ (1) $f_i \in Th^k(L)$, or (2) $f_i \in P$, or (3) f_i is the result of applying an inference rule to some members $f_{j_1}, \ldots, f_{j_m}(j_1, \ldots, j_m < i)$ of the sequence and $D_{\Rightarrow}(f_i) \leq j$. The set of all formulas which are j^{th}-degree-deducible from P based on $Th^k(L)$ is called the j^{th} *degree fragment of the formal theory with premises* P *based on* $Th^k(L)$, denoted by $T^j_{Th^k(L)}(P)$.

Automated reasoning is concerned with the execution of computer programs that assist in solving problems requiring reasoning. By adopting a suitable formal logic system for a target problem, we can do logically valid reasoning and get unknown or undecidable facts/hypotheses from empirical theorems that are well-known theories in a target domain. To do such logically valid reasoning automatically, a mechanism of automated reasoning is demanded. A forward reasoning engine is a computer program to automatically draw new conclusions by repeatedly applying inference rules to given premises and obtained conclusions until some previously specified conditions are satisfied. A facility to do reasoning automatically can be implemented by such forward reasoning engines and logic systems that are suitable for a target problem.

As a methodology of prediction, a method using anticipatory reasoning based on temporal relevant logics or 3D spatio-temporal relevant logics was proposed [38, 46]. Prediction is the action to make some future events known in advance, especially on the basis of special knowledge. It is a notion must relate to point of time to be considered as the reference time. For any prediction, both the predicted thing and its truth must be unknown before the completion of that prediction. An anticipatory reasoning is a reasoning to draw new, previously unknown and/or unrecognized conclusions about some future event or events whose occurrence and truth are uncertain at the point of time when the reasoning is being performed [38]. To represent, specify, verify and reason about various objects in the real world and relationships among them in the future, any ARRS needs a right fundamental logic system to provide a criterion of logical validity for anticipatory reasoning as well as formal representation and specification language. Temporal relevant logics and 3D spatiotemporal relevant logics are hopeful candidates of such right fundamental logic systems for ARRSs [38, 46]. Furthermore, to perform anticipatory reasoning automatically, an anticipatory reasoning engine was proposed and its prototype was implemented [44, 72, 132]. An anticipatory reasoning engine is a forward reasoning engine to perform anticipatory reasoning based on temporal relevant

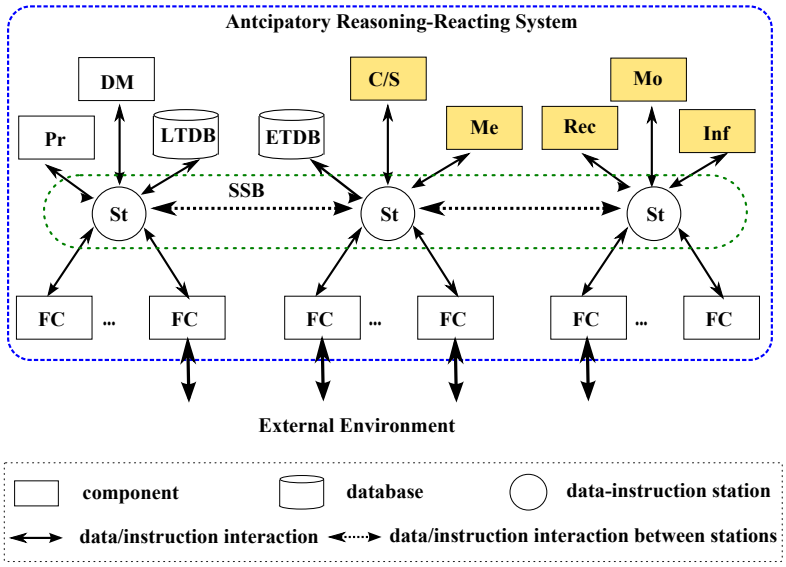

Figure 3.1: An architecture of an anticipatory reasoning-reacting system

logics or 3D spatio-temporal relevant logics.

On the other hand, a decision-making method with reasoning about actions was proposed [105, 106, 107]. An action in a computing anticipatory system is a deed performed by the system such that as a result of its functioning a certain change of state occurs in the system. To take next actions, at first, a computing anticipatory system enumerates all actions that the system can perform in a predicted future situation as candidates of next actions, and then, the system chooses appropriate actions as next actions to defend the system from possible failures and attacks. Reasoning about actions in a computing anticipatory system is the process to draw new conclusions about actions in the system from some given premises, which are already known facts or previously assumed hypotheses concerning states of the system and its external environment [106]. The decision-making method uses reasoning about actions to enumerate candidates of next actions. Deontic relevant logics and temporal deontic relevant logics are adopted as hopeful candidates of right fundamental logic systems for reasoning about actions [42, 106, 107]. Furthermore, to perform reasoning about actions automatically, an action reasoning engine was proposed and its prototype was implemented [106, 107]. Like the anticipatory reasoning engine, an action reasoning engine is a forward reasoning engine to perform reasoning about actions based on deontic relevant logics or temporal deontic relevant logics.

3.2 Overview of ARRS

The following content in the section is according to [70].

An architecture of an ARRS was proposed [71]. Figure 3.1 shows the architecture of the ARRS. An ARRS is a persistent computing system. A persistent computing system [4] can be constructed by a group of control components that

are independent of systems, a group of functional components that carry out special tasks of the system, and soft system buses (SSBs). Control components may include a central controller/scheduler (C/S), a central measurer (Me), a central recorder (Rec), a central monitor (Mo), and an central informant (Inf). A central controller/scheduler orders and controls all components to carry out some operations with a high priority. A central measurer measures current status of the system, and stores measured data into a central recorder. A central recorder stores data observed by a central measurer, and provides them to a central monitor and a central controller/scheduler. A central monitor monitors the behavior of the whole of the system, and reports unexpected behavior or troubles to a central informant. A central informant receives such reports from a central monitor, and informs the reports to managers of the system. A soft system bus is simply a communication channel with the facilities of data/instruction transmission and preservation to connect components in a component-based system. It may consist of some data-instruction stations (St's), which have the facility of data/instruction preservation, connected sequentially by transmission channels, both of which are implemented by software techniques, such that over the channels data/instructions can flow among data-instruction stations, and a component tapping to a data-instruction station can send data/instructions to and receive data/instructions from the data-instruction station. SSBs are used for connecting all components such that all data/instructions are sent to target components only through the SSBs and there is no direct interaction that does not invoke the SSBs between any two components.

Functional components of an ARRS are classified into two kinds components; ones are common components in all ARRSs and others are application-dependent components. A predictor (Pr), a decision maker (DM), a logical theorem database (LTDB), and an empirical theorem database (ETDB) are the common components. A predictor receives several kinds of data and outputs predictions with quantitative information and predictions without quantitative information. A decision-maker receives two kinds of predictions from a predictor, and outputs instructions to an ARRS. A logical theorem database stores fragments of logic systems underlying anticipatory reasoning or reasoning about actions. An empirical theorem database stores empirical theorems of a target domain as predictive models, behavior models, or world models, and empirical theorems deduced from the models by the predictor or the decision-maker. A predictive model is a set of empirical theorems and related with time in a target domain of the system. A behavior model is a set of empirical theorems and related with behavior in a target domain of the system. A world model is a set of empirical theorems of the target domain except empirical theorems related with time and behavior.

PCS-core components are control components and soft system buses. They are common in all persistent computing systems. ARRS-core components are a predictor, a decision-maker, a logical theorem database, and an empirical theorem database. They are common components in all ARRSs, but not in all persistent computing systems.

3.3 Safety and information security

There are two aspects of ARRS for ensuring the safety or information security.

First aspect is correctness. In ARRS, the correctness is depending on two layers. First layer is the correctness or suitableness of anticipatory model and measurement information. If they are correct, maybe the output of the system is correct. The second layer is the guarantee, that if the model is correct, and the measurement information is correct, then the output is correct. The correctness of prediction and decision-making is depending on logic system by adopting forward deduction, not abduction or induction. The feature of deduction is if the premise is correct, then the conclusion is correct. For the mechanism of prediction and decision-making, we adopt forward deduction based on strong relevant logic. Therefore, we can guarantee that.

The second aspect is performance. If the anticipatory actions of ARRS is given before the accident occurs/the consequence of attacks occurs, ARRS adopt suitable action to prevent these accidents/attacks.

3.4 ARRS: The candidate of target system for extension

ARRSs are suitable candidates of target systems for the extension, because (1) an ARRS uses logical reasoning to predict and make decisions, and (2) the architecture of ARRS is an extension of a reactive system.

An ARRS uses logical reasoning method to predict and make decisions [38], while such method can be applied to different reactive systems from various application areas. We can apply logical reasoning method to different types of reactive systems in different areas. The basic idea of logic-based reasoning method is to explicitly separate the underlying logical system, reasoning/computing mechanism, and empirical knowledge in any prediction/decision making, such that both underlying logical system and empirical knowledge can be revised/replaced/customized in various predictions/decision making processes performed by an area-independent, task-independent, general-purpose reasoning mechanism [44, 45]. The method of prediction/decision making by logic reasoning has the following characteristics [44, 45]. First, to explicitly separate the underlying logic system, reasoning/computing mechanism, and empirical knowledge directly results in an adaptive prediction/decision making method such that one can easily develop, modify, and improve the underlying logic system, reasoning/computing mechanism, and empirical knowledge separately to satisfy different requirements from various application areas. Second, as the underlying logic system (i.e., logical validity criterion) and empirical knowledge are separated explicitly, the prediction/decision making results can be evaluated from two aspects: logical aspect that is area/problem independent and empirical aspect that is area/problem dependent. Third, as a characteristic of development and maintenance technology, the underlying logic system, reasoning/computing mechanism, and empirical knowledge can be developed and maintained separately. Fourth, because logic systems and their formal lan-

guages are adopted as the absolute, area/problem independent criterion of correctness/validity and representation languages, the approach of prediction/decision making by logic reasoning is more suitable to qualitative methods rather than quantitative methods.

As an extension of a reactive system, the ARRS's architecture [71, 159] is fit to embrace different reactive systems. An ARRS consists of PCS-core, ARRS-core components, and application-dependent components, and connects all components using soft system bus (SSB) methodology [39]. SSBs methodology provides us to build persistent computing systems such that they can be easily maintained, upgraded, or recongured, without changing the basic system architecture, by adding some new components and/or data-instruction stations for satisfying new requirements, replacing an old or problematic component and/or datainstruction station with a new or sound one, and removing some useless components and/or data-instruction stations. The maintenance, upgrade, or reconguration of a persistent computing system based on SSBs can be done anytime by using the facility of data/instruction preservation of SSBs without stopping the running of the whole system [43]. We can view a reactive system or a component/module of a certain reactive system as an application-dependent component in ARRS. Therefore, an ARRS has the ability to embrace other reactive systems.

Chapter 4

A methodology to make existing reactive systems anticipatory

4.1 Overview

Our methodology aims to combine an existing reactive system, called *legacy system*, with some components which can provide anticipatory ability, then get a new system that in fact is an ARRS, called *target system*, which can deal with accidents/attacks anticipatorily with preserving the original system functions.

4.2 Requirements of legacy systems

As we discussed in section 2.6, the legacy reactive system must satisfy the following requirements.

- There must be some approaches to perceive adequate subset of the legacy system's the input set for detection and prediction.
- The legacy system must have some implemented mechanism to take actions to handle the ongoing/potential accidents/attacks.

Besides, for some extreme real time systems, it may be too late to predict and take actions before the accidents/attacks cause pernicious consequences, thus such systems are not suitable to extend with anticipation.

4.3 Target system

Figure 7.1 shows a general architecture of the target system embracing a legacy reactive system. Figure 7.2 shows the data flow diagram of the target system. Besides the legacy system, the target system includes following components.

There are three databases. *LTDB* is a logical theorem database, which stores fragments of logical systems [38]. *ETDB* is an empirical theory database storing *anticipatory model*, including world model, predictive model, and behavioral model,

which express the real world, predictive laws and behavioral patterns of the target domain as empirical theories represented by logical formulas correspondingly. *RDB* is a rule database storing *filter rules*, *translation rules*, *interesting formula definitions*, *interesting terms*, *actions mapping rules*, and *condition-action rules*.

Observers and filter deal with the data from the legacy system. *Observers* perceive the inputs of the legacy system that represent events and conditions produced by the environment. Besides, the observers could also perceive the status of the legacy system. In many cases, there are some sensors/monitors/statistical tools have been implemented in the legacy system. If we can utilize these observers, we do not need to implement new observers. *Filter* filters out the trivial sensory data and generates important and useful information for detection/prediction or decision-making.

Formula generator, forward reasoning engine, and formula chooser generate predictions or next actions. *Formula generator* encodes the sensory data into logical formulas used by the forward reasoning engine according to the translation rules. *Forward reasoning engine* is a program to automatically draw new conclusions by repeatedly applying inference rules, to given premises and obtained conclusions until some previously specified conditions are satisfied [47]. The forward reasoning engine gets logical formulas translated at the formula generator, fragment of (a) logic system(s), and the anticipatory model, and then it deduces candidates of predictions/next actions. *Formula chooser* chooses nontrivial conclusions from forward reasoning engine according to interesting formula definitions and interesting terms.

Action planner receives candidates of next actions and current situation, and then calculate the planned actions. The action planner has two functions: (1) the candidates of actions are only based on qualitative information of the situation, while the action planner can utilize quantitative information to revise the actions, and (2) to make accurate plan, calculation is needed, such as to calculate how many special actions should be taken to prevent a certain accident/attack.

Enactor receives planned actions from action planner, matches the current situations to the situation-action rules to select appropriate actions to take, and then gives corresponding instructions according to the actions mapping rules to invoke the actions of the legacy system. Besides, if the original reactive actions of the legacy system may make against the safety or security of the system in some situation, the enactor are also in charge of canceling these original actions. Condition-action rules specify when an unexpected accident/attack occurs abruptly in real time, (1) which reactive actions should be taken, and (2) which planned actions should be canceled.

Soft system bus [39] and *central control components* provide a communication channel for each component and an infrastructure for persistent computing [39].

4.4 Phases

In our methodology, there are five major phases to extend a legacy reactive system with anticipatory ability.

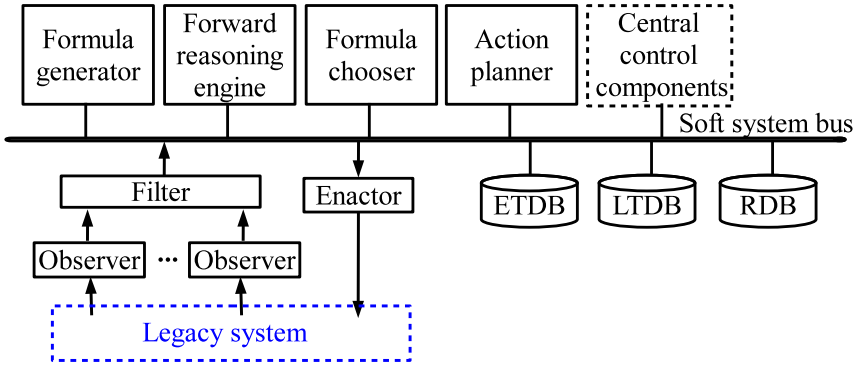

Figure 4.1: A general architecture of the target system

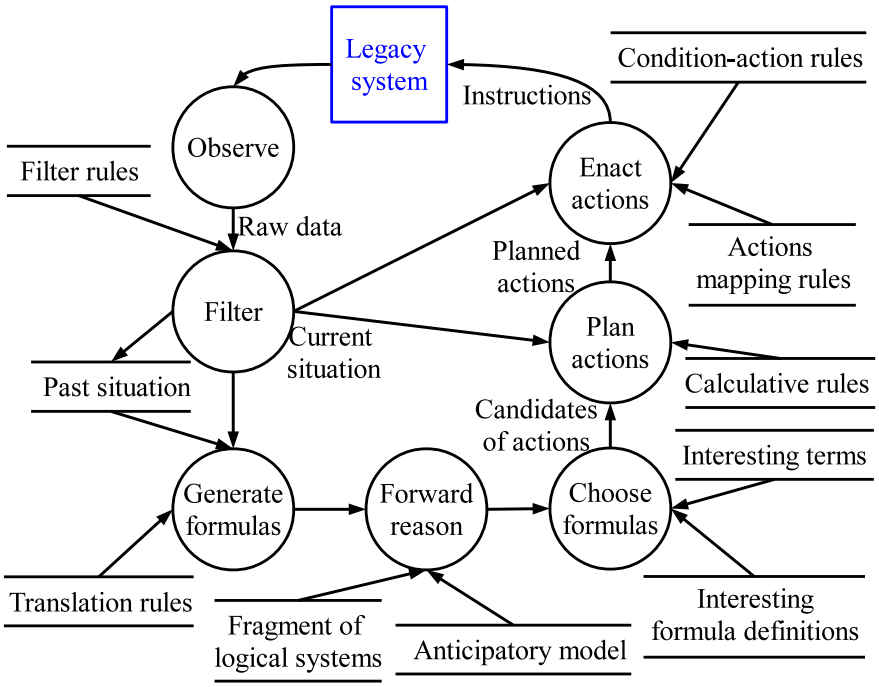

Figure 4.2: Data flow diagram of the target system

Phase 1: Analyze the target domain

First, we analyze the possible accidents/attacks in the legacy system's target domain. We list possible accidents/attacks in the target domain, and then for each possible accident/attack. We analyze each possible accident's/attack's causation, formation process, and the consequence, as well as other attributes such as likelihood and severity. Based on the severity and likelihood, we could sort these possible accidents/attacks in order of importance.

Second, we analyze the possible (anticipatory) actions for defending against accidents/attacks in the target domain. For each possible accident/attack, we find out whether it is possible to defend against it and what actions could defend against it, both an ongoing one and a potential one.

Phase 2: Analyze the legacy system

First, we need to understand the requirements of the legacy system. Second, for the legacy system, we list the set I of inputs that represent events and conditions produced by the environment and the set O of output action instructions produced by the legacy system, and understand the function f_P of the legacy system. Third, we find out which anticipatory actions the legacy system has already had the ability to take, by comparing the outcome of phase 1.2 and the set O of outcome of phase 2.2. Fourth, we analyze the legacy system to find out which ongoing/potential accidents/attacks the system already has the ability to handle, and how the system defends against them. After that, we could also find out which accidents/attacks that the legacy system cannot handle or do not handle well, which may be the goals of the target system. Fifth, we find out the approaches to perceive the inputs that represent events and conditions produced by the environment and (optional) status of the legacy system itself, and the approaches to invoke these existing anticipatory action instructions of the legacy system.

Phase 3: Define requirements of the target system

The requirements of target system include the functional requirements that specify which accidents/attacks should be tackled, and which anticipatory actions should be used to prevent against a certain accidents/attacks, and the non-functional requirements including performance, security, availability, etc.

Phase 4: Construct anticipatory model

First, before constructing anticipatory model, we need to choose the logic basic(s) for the target domain. A model is a simplified representation of something [52]. In the target system, an anticipatory model is represented by a set of logical formulas of a specific logic (or logics). Thus before we build an anticipatory model, we must choose a logical basis for the model. Such logic must satisfy the following requirements [38]. First, the logic must be able to underlie relevant reasoning as well as truth-preserving reasoning in the sense of conditional. Second, the logic must be able to underlie ampliative reasoning. Third, the logic must be

able to underlie paracomplete and paraconsistent reasoning. Especially, the logic for prediction must be able to underlie temporal reasoning.

Second, we construct world model. A world model represents status of the real world and essential empirical knowledge (not related with time and behavior) in target domain. We use a vocabulary of predicates or constant terms to present the status of the real world. In addition, if necessary, we collect and represent empirical knowledge (not related with time and behavior) as conditionals (see "conditional 2" [52]). The construction of the world model involves following steps. (1) In the target domain, list possible objects and their properties and statuses. (2) List relationships among the objects (if have). (3) List possible events and conditions produced by the environment we concern. (4) Determine the essential empirical knowledge (not related with time and behavior). (5) Formalize the information. For information got from step (1) to (3), we use a vocabulary to represent, while for information got from step (4), we use conditionals to represent.

Third, we construct predictive model. The predictive model represents the predictive knowledge used to make predictions. We are only interested in certain kinds of predictions, while in the target system, they are mainly accidents/attacks and other predictions, which can help to predict accidents/attacks. Therefore, the predictive model is assembled by conditionals, which can be used to get these interesting predictions. The construction of the predictive model involves following steps. (1) Determine which events/conditions (accidents/attacks and other predictions, which can help to predict accidents/attacks) we concern in the target domain, and list these events/conditions. (2) Determine the predictive knowledge related with these events/conditions. (3) Formalize the knowledge as conditionals.

Fourth, we construct behavioral model. The behavioral model is used for qualitative decision, which represents the behavioral knowledge used to choose actions. The purpose of qualitative decision is to find out "what actions should be taken?" and "which actions should be taken first?" Thus, the result of qualitative decision is a set of candidates of next actions, which are labeled as "obligatory"/"permitted" and/or priorities. Therefore, the behavioral model is assembled by conditionals, which can be used to get these results of qualitative decision. The construction of the behavioral model involves following steps. (1) List possible actions that the system can be taken to prevent accidents/attacks. (2) Decide which events/conditions cause to take these actions. (3) Formalize the information. For information got from step (1), we use a vocabulary to represent, while for information got from step (2), we use conditionals to represent.

Fifth, we evaluate the constructed anticipatory model. To evaluate anticipatory model, we could utilize some (user-defined) evaluation criteria to evaluate the model formally. Besides, in practical, we can use test set for evaluation, which is a set of empirical/historical data about accidents and attacks in the target domain.

Phase 5: Implementation

Most components of the target system are general-purpose. Thus, we do not need to rebuild them for different systems. There are some implemented components can be used in this phase, such as a general-purpose forward reasoning

engine [47]. We have also implemented a general filter, a semiautomatic formula generator, a general formula chooser and a general enactor, which may deal with most cases. Our methodology considers using these prepared components. Of course, one can rebuild all of these components.

First, we implement observers or utilize some substitutes in the legacy system to observe events and conditions produced by the environment, which relate to requisite events/conditions used by anticipatory model. Second, we configure filter rules to filter nontrivial events/conditions used by anticipatory model. Third, we configure the actions mapping rules. The mapping rules map the possible anticipatory actions (based on behavioral model) are represented as logical formulas to concrete instructions that the target system can execute. Fourth, we configure translation rules according to the logic basis and anticipatory model. Fifth, we define the interesting formulas and configure the interesting terms. Interesting terms specify which terms we are interested: (1) for detection/prediction, the interesting terms are events and conditions produced by the environment about accidents and attacks, and (2) for decision-making, the interesting terms are the actions, which can be produced by the system. Interesting formula definitions specify which formulas, formalized by interesting term(s) and other symbols, we are interested. Sixth, we implement action planner and configure calculate rules. Not all target systems need quantitative calculation. Because quantitative calculation depends on the application domain, we may resort to different algorithms thus build different action planners for different target systems. Seventh, we implement the enactor and configure condition-action rules. Eighth, we integrate all components. Ninth, we test the target system. Tenth, we evaluate the target system before putting into service.

Chapter 5

Case study: Emergency elevator evacuation systems

5.1 Overview

Although current *emergency elevator evacuation system* (EEES) have made big progress in their physical safety, these systems are still in an early phase for evacuation elevator control, because the current EEESs (1) cannot automatically deal with different evacuation types, different extraordinary events, and cannot dispatch the elevator cars according to the evacuation type, the extraordinary event, and the current situation, and (2) does not consider anticipation. In this case study, we applied the methodology to add anticipatory ability to the current emergency elevator evacuation systems with some sensory systems. After the extension, we got an anticipatory emergency elevator evacuation system (AEEES), which can predict and detect hazards in the emergency evacuation, and then dispatch the elevator cars anticipatorily, aiming to avoid hazards beforehand, thus to rescue more occupants and to shorten the evacuation time. In order to evaluate AEEES, we implemented a simulation program, which simulates the elevator systems and its environment, and then we took three groups of experiments to compare different EEESs: EEES using down-peak, reactive EEES, and AEEES. We considered the rescued ratio as the evaluation factor, and AEEES always got highest rescued ratio in the experiments.

5.2 Introduction

Using elevators for emergency evacuation has become reasonable and necessary for modern high-rise buildings in recent years. First, it is impossible to evacuate an ultra high-rise building in a tolerable time by only using stairs [31]. Using elevators can decrease evacuation time and reduce congestion on stairs which equates to less potential for injuries [51]. Second, the aged and people with disabilities or injuries can hardly use stairs to evacuate [31]. Third, fire is not the only reason for evacuation. Sometimes it is important to leave the building before a situation gets worse, such as a bomb threat or other acts of terrorism [51]. Lastly, the current

elevator systems can provide safe and reliable operation both for fire service access and for occupant egress during fires [51, 113].

An *emergency elevator evacuation system* (EEES) includes the elevator equipment, hoistway, machine room, and other equipment and controls needed for safe operation of the elevator during the evacuation process [109]. Because the purpose of emergency evacuation is to move people away from the threat or actual occurrence of a hazard immediately and rapidly, an ideal EEES should have the following features: safe, context aware, anticipatory, and instructional/informative. However, the current researches of EEESs mainly focus on the physical safety of the elevator systems; few researches involve context aware of EEESs; let alone anticipation.

As an attempt to realize an advanced EEES with safe, context aware, anticipatory, and instructional/informative features, we propose a new type of EEESs, named *anticipatory emergency elevator evacuation system* (AEEES), which can detect and predict hazards in the emergency evacuation, and then dispatch the elevator cars anticipatorily aiming to avoid hazards beforehand, thus to rescue more occupants and to shorten the evacuation time. In order to implement the AEEES, we adopted our methodology to extend current EEES to be an ARRS, indeed an AEEES. To evaluate the AEEES, we present experiments for comparison the basic EEES which dispatch elevator cars using down-peak [23], the reactive EEES with context awareness but without anticipation, and our proposed AEEES. However, we do not discuss the physical safety of the EEES, because there are a lot safety elevator have been practically used [31].

5.3 Ideal emergency elevator evacuation systems

What should an ideal *emergency elevator evacuation system* (EEES) be? Because the purpose of emergency evacuation is to move people away from the threat or actual occurrence of a hazard immediately and rapidly, an ideal EEES should have the following features.

Safe: The EEES must ensure the safety of the passengers who take the elevators during the emergency evacuation.

Context aware: The EEES can autonomically both sense and react against the particular case of emergency. "Sense" means the EEES can detect the emergency events (e.g., fire, gas leaks), perceives the current situation (e.g., which rooms are catching fire in a fire emergency, an elevator car is full loaded), and recognizes special people (e.g., people with disabilities or injuries). "React" means the EEES decides which evacuation type is taken according to the emergency situation (including total evacuation, staged evacuation, and fractional evacuation [51]), and reacts to some particular events (e.g., not stopping the elevator car in fire, dispatching a fully loaded elevator directly to the evacuation floor).

Anticipatory: In order to indeed make "people away from the threat", the EEES should be anticipatory, i.e., the EEES should predict the future situations (e.g., who will be in danger soon, and there will be a congestion in some stairs), then dispatch the elevator cars beforehand, aiming to avoid disaster beforehand

and shorten the evacuation time. A serious emergency usually gets worse rapidly with deadly damage. For example, in an uncontrollable conflagration, because the fire spread with great rapidity, it is too late to egress when fire and poisonous smoke draw near. Thus the EEES should have the ability to predict which area of building will be dangerous, then rescue the people in that area. Besides, if the EEES can predict where a congestion will happen, then transport some people in that area, thus a trample may be avoided.

Instructional/informative: The EEES can instruct occupants to use the elevators to evacuate, and inform occupants the current situation of emergency by public address or other approaches. Besides, the EEES must instruct occupants which elevators can be used, who should use the elevator first, as well as inform occupants the emergency situation, and how long they have to wait the next shift.

5.4 Current emergency elevator evacuation systems

The current researches of emergency elevator evacuation systems (EEES) mainly focus on the physical safety of the elevator systems, and there is still a gap between current EEESs and ideal EEESs. Klote et al. studied the feasibility of elevator evacuation of FAA air traffic control towers and developed the concept of EEES [109]. The researches of EEESs can be divided two categories: physical safety and elevator cars dispatch for emergency.

Physical safety is the basic challenge of EEESs, i.e., to protect elevators from heat, flame, smoke, water, overheating of elevator machine room equipment, loss of electric power, as well as to assure the safety of people traveling in the elevators. Several researches were carried out on this topic [113, 109, 110]. As a result, several kinds of elevators for evacuation were developed, which can provide the above safety features, such as the enhanced elevators and protected elevators [51]. By now, some skyscrapers have equipped these elevators for evacuation [31].

In addition to safety, another challenging problem for EEES is how to control elevators autonomically, effectively and safely according to the current evacuation type and extraordinary event. Until now, there are mainly following elevator control strategies for evacuation.

Neglect: Some EEESs do not provide additional control strategies for evacuation. These systems just use general elevator algorithm or general group control algorithm for evacuation.

Manual control and semiautomatic control with human oversight [32, 121]: Large buildings often have a command center for directing an emergency evacuation. Operators in command center can get information from alarm systems, floor monitors, television or security cameras, then dispatch elevators to where they are needed manually. Operators can also use some automated control system to set priorities and determine which floors should evacuate using the elevators.

Down-peak and its variation: In down-peak mode, elevator cars depart from the lobby to the highest floor served, then run down the floors in response to hall calls placed by passengers wishing to leave the building. There are also some im-

proved group elevator down-peak scheduling (including zoning of elevator groups) for emergency evacuation [123]. For evacuation, down-peak mode is proved more better than other special operating modes such as up-peak [23]. The benefit of down-peak is easy to implement.However, only using down-peak cannot consider different emergency evacuation types, such as fractional emergency evacuation [51], and current perils and situation of the scene.

Dispatching elevators based on the current situation: Some patents claimed their systems can measure the number of people remaining inside a building, detect an emergency condition in the building, then dispatch the elevators according to these current situation [100, 101, 137]. However, these patents do not show how to fulfill their claim in detail. Moreover, these systems can only dispatch elevators based on certain kinds of information, but cannot deal with complex situation dynamically.

Theoretical evacuation strategies: There are some theoretical elevator evacuation strategies were proposed, that can be used to determine which floor should be evacuate firstly [30, 73, 149]. However, they are only theoretical work and do not show how to use these strategies in a practical EEES.

In summary, although the current EEESs have made big progress in their physical safety, the current EEESs are still in an early phase for autonomic evacuation elevator control, because the current EEESs (1) cannot automatically deal with different evacuation types, different extraordinary events, and cannot dispatch the elevator cars according to the evacuation type, the extraordinary event, and the current situation, and (2) does not consider anticipation.

5.5 Anticipatory emergency elevator evacuation systems

We propose a new type of EEESs, named *anticipatory emergency elevator evacuation system* (AEEES). An AEEES can automatically detect and predict hazards in the emergency evacuation, and then dispatch the elevator cars anticipatorily, aiming to avoid disasters beforehand, thus to rescue more occupants and to shorten the evacuation time. Because the physical safety of AEEES is ensured by the enhanced/protected elevators, the heart of AEEES is an information system to control elevator cars and to instruct occupants. Thus, we only focuses on the information system, but not the whole of AEEES as well as the physical safety.

In order to fulfill the features of context aware, anticipatory, and instructional/-informative, we analyzed the system requirements of the AEEES as follows.

R1: The AEEES should perceive the current situation of the building by utilizing the sensory data from different kinds of sensors and monitor systems.

R2: The AEEES should deal with different type of emergency, such as fire, gas leaks, bomb threats, as well as several types of emergency happens in the same time. Besides, the system should deal with new type of emergency with trivial modification.

R3: The AEEES should decide the evacuation type according to the current emergency, including total evacuation, staged evacuation, and fractional evacuation [51].

R4: The AEEES should predict the future hazards automatically and autonomically .

R5: The AEEES should automatically generate anticipatory actions (against the future hazards), reactive actions (against the ongoing hazards), and routine actions (for total occupation, e.g. down peak). Besides, the AEEES should judge which actions should be taken first automatically.

R6: The AEEES should inform the corresponding occupants the ongoing hazards and the predictions of hazards, and which elevators can be used and when next shift comes, as well as instruct the occupants what to do next. The content of the messages are chosen automatically. The AEEES gives messages by public address system or other communication systems.

R7: The AEEES should know evacuation for certain type of emergency should stop automatically, when that emergency was eliminated (e.g. the fire dies out).

R8: The AEEES should be portable for different buildings with trivial modification, such as different building structure, different elevator systems, different sensors, and different layout of elevators and sensors.

5.6 Simulation program of the legacy system

Because of cost considerations, we did not consider applying our methodology to a real elevator system, and instead, we implemented a simulation program, which simulates the elevator system and its environment (e.g., building, floors, elevator lobby, occupants, and emergencies). The simulation program provides the interface to observe the situation, the interface to control the elevator cars, the interface to implement different emergencies, and the statistic module, thus we can implement and compare different EEESs based on the simulation program. The simulation program reserves abundant adjustable settings, e.g., number of floors of the building, number of elevator cars, the layout of elevators, elevator properties (e.g., speed, time for stop, time for entrance/exit), number of occupants for each floor, occupants with disabilities. In this simulation program, we assume the elevator car and the evacuation floor is aways safe, and if the elevator lobby is in emergency, and then people cannot egress from corresponding elevator. We also implemented some emergencies, such as fire (based on [130, 151]) and biological/chemical leak emergency. Besides, each emergency has adjustable settings, e.g., spread speed for different directions, and acceleration. The simulation program also provides a graphic display to show the whole progress of evacuation, as well as the emergencies.

5.7 Applying the methodology

In this section, we use each phase of the methodology to extend the legacy systems to be an AEEES.

Phase 1: Analyze the target domain

The possible accidents are disasters or bad situations, such as fire, toxic fume, biological/chemical leak, congestion, etc. The possible anticipatory actions for an EEES is to dispatch the elevator cars anticipatorily, aiming to avoid disaster beforehand and shorten the evacuation time.

Phase 2: Analyze the legacy system

The set I includes: (1) the occupants' direct operations to the elevator system, (2) the emergency events, such as fire, bomb threats, biological or chemical threat, and (3) occupants' conditions, such as where the occupants and occupants with disabilities or injuries are. For O, the actions are mainly dispatching the elevator cars, furthermore, some elevator cars also support "shuttle mode", to avoid time needed for accelerating and decelerating smoothly [30], besides, the public address system can give warnings and instructions to occupants. The anticipatory actions are dispatching the elevator cars. The legacy system only ensures the safety of elevator cars and evacuation floors, but cannot dispatch the elevator cars anticipatorily. For I, there are many existing equipments can perceive the status of the building (including elevator systems and occupants) and occupants. Emergency sensors (heat detectors, smoke detectors, flame detectors, gas sensors, sensors to detect chemical and biological leak, etc.) can detect emergencies. The status of occupants, such as number of occupants in each floor/area, and whether there are disabled or injured people, can be monitored by using RFID [123] or cameras [103]. We can connect the filter directly to these sensors by invoking the sensors' driver program. For O, we can dispatch the elevator cars by giving instructions to elevator control system of the legacy system. When an emergency arises, the elevator control system could disable the regular dispatch mode, and receives instructions from enactor.

Phase 3: Define requirements of the target system

Because the purpose of emergency evacuation is to move people away from the threat or actual occurrence of a hazard immediately and rapidly, we consider the target EEES should be safe, context aware, anticipatory, and instructional/informative.

Phase 4: Construct anticipatory model

We choose *temporal deontic relevant logics* [42] as the logic basis to both predict and make decisions. Such logics satisfied the requirements of logic basis of phase 4.

Table 5.1: Predicate dictionary of the world model

Predicate	Meaning
$Floor(f)$	f is a floor
$Lobby(l)$	l is an elevator lobby
$Upstairs(f_1, f_2)$	f_1 is upstairs of f_2
$Locate(l, f)$	elevator lobby l locates in floor f
$Elevator(e)$	e is an elevator car
$Getatable(e, l)$	elevator e is getatable to lobby l
$Occupied(r)$	region r has alive people
$Afire(r)$	region r is afire
$Smoke(r)$	region r is full of smoke
$Poison(r)$	region r has biological or chemical threat
$Person(o)$	o is a person
$Priority(o, p)$	o has priority p for rescuing
$In(o, r)$	o in the region r
$Eliminated(et)$	emergency et has been eliminated

Table 5.2: Predicate dictionary of the behavioral model

Predicate	Meaning
$Pickup(f, p, et)$	pick up people on floor f with priority p due to emergency type et
$FullEvacuation(et)$	adopt full evacuation from the building because of emergency type et
$StopEvacuation(et)$	stop the evacuation because of emergency type et
$Dispatch(e, f, p, et)$	an action that elevator e to pick up people on floor f with priority p because of emergency type et

In this case, we build the vocabulary as a predicate dictionary, shown in table 5.1. Moreover, we do not need other empirical knowledge in the world model.

In this case, the interesting predictions are emergencies. There are different empirical theories for different emergencies. For example, in fire emergency, to predict fire spread, we have following empirical theorem "the upper floor of the afire floor will catch fire with highly probable" [130, 151]: $\forall f_1 \forall f_2 (Upstairs(f_1, f_2) \wedge Afire(f_2) \wedge \neg Afire(f_1) \Rightarrow F(Afire(f_1)))$. To predict based on prediction, we have: $\forall f_1 \forall f_2 (Upstairs(f_1, f_2) \wedge F(Afire(f_2)) \wedge \neg Afire(f_1) \Rightarrow U(Afire(f_2), F(Afire(f_1))))$, which means "if a floor will catch fire, and then when this floor is really afire, its upstairs will catch fire".

Because we use temporal deontic relevant logics as the logic basis, we use "O" to specify an action is "obligatory" to do, and "P" for "permission". In order to express more precision, we also introduces priority constants (using floating number to express, smaller means higher priority): $CRITICAL > HIGH > MEDIUM > NORMAL > LOW > PLANNING$. As shown in table 5.2, we build the vocabulary of actions (only the last one is an explicit instruction to dispatch the elevator cars). There are three types of actions in an AEEES:

reactive, anticipatory, and routine. Reactive actions deal with current crises. Anticipatory actions deal with predictive crises. Routine actions are used in a full evacuation to egress all occupants in the building, e.g., down peak. Now we show how to construct the behavioral empirical knowledge. For example, in fire emergency, "the occupants of the fire floor are the occupants at the highest risk and should be the ones to be evacuated first by the elevators" [149], which can be expressed as empirical theorem about reactive actions: $\forall f(Afire(f) \wedge Floor(f) \wedge Occupied(f) \Rightarrow O(Pickup(f, CRITICAL, FIRE)))$. To predict crises, we have empirical theorems about anticipatory actions: $\forall f(F(Afire(f)) \wedge Floor(f) \wedge Occupied(f) \Rightarrow O(Pickup\ (f, HIGH, FIRE)))$ and $\forall f \forall a(U(a, F(Afire(f))) \wedge Floor(f) \wedge Occupied(f) \Rightarrow O\ (Pickup(f, MEDIUM, FIRE)))$. To express routine actions, e.g., down peak can be expressed as: $\forall et \forall f(FullEvacuation(et) \wedge Floor(f) \wedge Occupied(f) \Rightarrow P(Pickup\ (f, PriorCal(NORMAL, f), et)))$, which means when we want to apply full evacuation because of emergency type et, we can pick up each floor with priority $PriorCal(NORMAL, f)$. $PriorCal$ is a function to calculate priority, which result is $NORMAL - 0.001 \times FloorNumber(f)$, meaning higher floor has higher priority. To consider full evacuation when fire emergency, can be expressed as empirical theorem: $\forall r(Afire(r) \Rightarrow FullEvacuation(FIRE))$. Besides, because the aged and people with disabilities have higher priority in any emergency, we have: $\forall o \forall p \forall f(Person(o) \wedge Priority(o, p) \wedge Floor(f) \wedge In(o, f) \Rightarrow P(Pickup(f, p, ANY)))$. Because the elevator cars are the executor of the actions, we use $Dispatch(e, f, p, et)$ to express the action elevator e to pick up people on floor f with priority p because of emergency type et. We have: $\forall f \forall p \forall et \forall e \forall l(O(Pickup(f, p, et)) \wedge Elevator(e) \wedge Getatable(e, l) \wedge \neg IsFire(l) \wedge Locate(l, f) \Rightarrow O(Dispatch(e, f, p, et)))$ and $\forall f \forall p \forall et\ \forall e \forall l(P(Pickup(f, p, et)) \wedge Elevator(e) \wedge Getatable(e, l) \wedge \neg IsFire(l) \wedge Locate(l, f) \Rightarrow P(Dispatch(e, f, p, et)))$. When certain emergency was eliminated (e.g. the fire dies out), the AEEES should know the evacuation should stop, thus we have: $\forall et(Eliminated(et) \Rightarrow P(StopEvacuation(et)))$.

In this case, we integrate the evaluation of anticipatory model into the experiment in section 5.8.

Phase 5: Implementation

Because we choose temporal deontic relevant logics as the logic basic, for prediction, the interesting formula (IF) is defined as: 1) If A is an interesting term, then A is an IF; 2) If A is an interesting term, then $\neg A$ is an IF; 3) If A or B is IFs, then $U(A, B)$ is an IF; 4) If A is an IF, then $\forall x A$ and $\exists x A$ are IFs; 5) If A is an IF, then ΦA is an IF, where Φ is one of $\{G, F\}$. For candidates of actions, the IF is defined as: 1) - 3) are as same as that of prediction; 4) If A is an IF, then ΦA is an IF, where Φ is one of $\{O, P\}$. For the prediction of fire, $Afire$ is an interesting term. For candidates of actions, $Pickup$, $FullEvacuation$, $StopEvacuation$, and $Dispatch$ are interesting terms.

The action planner can refine the actions by quantitative calculation, in order to plan precise elevator cars dispatch. In this case study, the action planner also takes charge of canceling actions when certain emergency was eliminated. The action planner outputs next planned actions for each elevator car. The action planner

maintains a two level priority set called *action_container*, where the low priority subset stores all candidates of actions containing predicate P, the high priority subset stores remaining candidates of actions, and each subset sorts candidates of actions according to their priority. When we fetch elements from the two-level priority set, we first fetch the actions with highest priority from the high priority subset, after the high priority subset has been traversed, we fetch the actions with highest priority from the low priority subset. The action planner maintains a variable called *planned_action* for each elevator cars. An algorithm to calculate quantitative *planned_action* is:

```
for each (elevator e1 in all elevators){
  if (e1 is free){
    for each (action in action_container){
      planRescuePeople = 0;
      for each (elevator e2 in all elevators)
        if (e2.planned_action.floor
            == action.floor)
          planRescuePeople += e2.CAPACITY;
      if(action.floor.peopleNumber+REDUNDANTNUM
           >planRescuePeople){
        e1.planned_action = action;
        break;
      }
    }
  }
}
```

The calculation is triggered by the event when an elevator car is free. When the elevator car arrives at the floor of its *planned_action*, we set its *planned_action* as $NULL$. We have following condition-action rule in the target system: "Never stop the elevator car at an elevator lobby with an ongoing emergency."

Furthermore, we can realize the instructional/informative feature by using the public address system to broadcast the current emergencies, predictions of emergencies, planned elevator dispatch, which are represented as logical formulas such that they can be easily translated into natural language.

5.8 Simulation experiments

To evaluate AEEES, we also implemented other two EEESs: EEES using down-peak and reactive EEES (to rescue the floor with emergency first, then using down-peak).

The basic experiment scenario is that a building with 25 floors (almost all skyscrapers zone elevators with sky lobby, which is equivalent to superposition of several short building) and 8 elevators, uniform distribution of occupants, and the emergency is uncontrollable accelerated fire with random origin of fire. The experimental parameters' setting of fire emergency is based on [151, 130], and other

parameters' setting is based on [108]. All occupants evacuate only by using elevators. We designed three groups of experiments: (1) origin of fire in different floor, 3,000 occupants in the building, (2) different fire spread speed, 3,000 occupants in the building, and (3) different number of occupants; while other parameters stay the same.

We consider the rescued ratio as the evaluation factor.

$$Rescued\ ratio = \frac{the\ number\ of\ rescued\ occupants}{the\ number\ of\ total\ occupants\ in\ the\ building} \times 100\%.$$

A good evacuation gets high rescued ratio.

Figure 5.1 shows the results of three groups of experiments. Using down-peak got lowest rescued ratio; AEEES got highest rescued ratio; and reactive EEES's rescued ratio was in between of them. To explain why such experimental results occur, we present a certain situation of first group of experiments. A fire emergency originated from 17 floor (F). Due to the fire alarm, all occupants began to evacuate from the building by using elevators. For down-peak, all 8 elevator cars were dispatched to 25F first. Because all occupants used elevators to evacuate, the car would stop at a floor which had occupants. However, the occupants threatened by the fire (might later die), such as occupants in 17F, cannot use the elevators in time. For reactive EEES, it sensed emergency on 17F, then dispatched all 8 cars to 17F first. Because there were 120 occupants in each floor and the maximum capacity of car is 20, these non-overloaded cars would stop at 16F and rescue occupants. (If a car is full, both reactive EEES and AEEES dispatch the car to evacuation floor without stopping.) After all occupants of 17F were rescued (the 18F was not afire at the moment), the reactive EEES adopted down-peak, thus all cars were dispatched to 25F. Later the 18F caught fire, reactive EEES sensed that and tried to rescue occupants on 18F. However, all cars were occupied at the moment, besides, the fire was severe, and there was little time left to rescue the occupants on 18F before they died. For AEEES, it filtered out the emergency on 17F and the situation of occupants, then transformed them into logical formulas (e.g., $Afire(17F)$, $Occupied(17F)$). Based on the predictive model and the world model (e.g. $Floor(17F)$, $Floor(18F)$, $Upstairs(18F, 17F)$), AEEES deduced candidates of predictions. Then AEEES chose $F(Afire(18F))$ and $U(Afire(18F), F(Aire(19F))$, which means 18F would catch fire, and after that 19F would catch fire. For the qualitative decision, the AEEES chose $Pickup(17F, CRITICAL, FIRE)$, $Pickup(18F, HIGH, FIRE)$, $Pickup(19F, MEDIUM, FIRE)$, $Pickup(1F, PriorCal(NORMAL, 1F), FIRE))$, ..., and $Pickup(25F, PriorCal(NORMAL, 25F), FIRE))$, which means to rescue occupants on 17F with CRITICAL priority, 18F with HIGH priority, 19F with MEDIUM priority, and other floor with NORMAL priority besides higher floor has higher priority. Based on the calculation algorithm, the AEEES first dispatched 7 elevator cars to 17F and 1 elevator cars to 18F, then dispatched 6 cars to 18F and 2 cars to 19 floor, next dispatched 5 cars to 19F, after that dispatch 7 cars to 25F and 1 car to 24F.

We noticed sometimes anticipation becomes invalid when the emergency is too severe or there are too many occupants. Its essence is when carrying capacity of elevators is insufficient, AEEES can only rescue occupants in emergency floors,

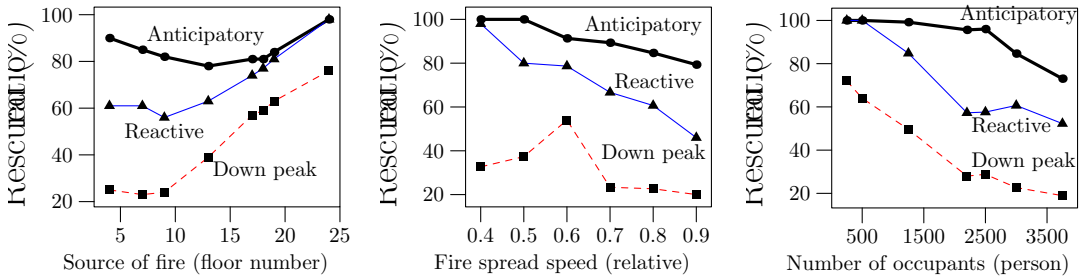

Figure 5.1: Results of three groups of experiments

thus AEEES degenerates to reactive EEES.

5.9 Summary

We presented anticipatory emergency elevator evacuation system (AEEES), its features, design, prototype implementation by extending existing EEES, and evaluation. As a new type of emergency elevator evacuation systems (EEES), AEEESs show a new direction for EEESs.

We will consider the behavior of occupants, emergencies' accuracy, various emergencies, and evaluation from the viewpoint of efficiency in the future work. In fact, occupants may use stairs to evacuate, and they may spend a long time before they take evacuation [148], thus AEEESs should deal with this. One urgent issue is that our system do not consider the smoke of fire which is more deadly than flame. In order to construct good predictive model for AEEESs, we need qualitative experimental knowledge about emergencies. However, the current researches about emergencies are mainly numerical. It is a challenge to extract qualitative knowledge from these quantitative experimental knowledge.

Chapter 6

Case study: Runway incursion prevention systems

6.1 Overview

Although current runway incursion prevention systems have made big progress on how to obtain accurate and sufficient information of aircraft/vehicles, they cannot predict and detect runway incursions as early as experienced air traffic controllers by using the same surveillance information, and cannot give explicit instructions and/or suggestions to prevent runway incursions like real air traffic controllers either. In this case study, we applied our methodology to existing airport/on board systems/equipment to build an ARRS, called anticipatory runway incursion prevention systems (ARIPS). The ARIPS predicts and detects runway incursions, then gives explicit instructions and/or suggestions to pilots/drivers to avoid runway incursions/collisions. The features of ARIPS include long-range prediction of incidents, explicit instructions and/or suggestions, and flexible model for different policies and airports. To evaluate ARIPS, we built a simulation system, and evaluated ARIPS using both real historical scenarios and conventional fictional scenarios of NASA. We did three groups of experiments to evaluate the performance, correctness, and generality of ARIPS. The evaluation showed that our system is effective at providing earlier prediction of incidents than current systems, giving explicit instructions and/or suggestions for handling the incidents effectively, and customizing for specific policies and airports using flexible model.

6.2 Introduction

Runway incursions are occurrences at an aerodrome involving the incorrect presence of an aircraft, vehicle or person on the protected area of a surface designated for the landing and take-off of aircraft [135]. Runway incursions have the potential to cause serious accidents with significant loss of life. The world's deadliest aviation accident was the result of a runway incursion [63]. In the United States, an average of three runway incursions occur daily [63]. Avoiding runway incursions has been a top ten priority for the National Transportation Safety Board for over

a decade [157].

The causes leading to runway incursions include pilot deviations, operational errors/deviations, and vehicle/pedestrian deviation [157]. All these causes belong to human factors.

Traditionally, to prevent runway incursions, both pilots and controllers rely on visual cues, occasional communications by radio, and their memories [182]. In recent years, people invented several modern devices and systems to decrease the chance of runway incursions [182]. Because all causes of runway incursions belong to human factors, runway incursion prevention systems require removing human from the system operation loop as much as possible [157]. Therefore, ideal runway incursion prevention systems should replicate the abilities of experienced air traffic controllers, i.e., the system should not only predict and detect an runway incursion but also give explicit instructions and/or suggestions to the pilots/drivers to avoid the runway incursion/collision.

Although several runway incursion prevention systems have been developed, current systems are only assistant tools for human, and still far from ideal. First, although current systems have made big progress on obtaining accurate and sufficient information of aircraft/vehicles, current systems cannot always predict/detect a runway incursion as early as an experienced air traffic controller predict/detect by using the same surveillance information. Second, current systems lack the ability to give explicit instructions and/or suggestions to prevent runway incursions. We consider a runway incursion prevention system should have such an ability, because (1) although alert of incursion is given, both the air traffic controllers and pilots/drivers need time to make decisions about instructions to prevent the incursion, and (2) at a critical moment, human may make mistakes due to overstrain, but machine do not. Third, current systems do not consider the special conditions of a certain airport and different areas of that airport. In contrast, experienced air traffic controllers can use such information for better prediction/detection.

In order to remove human factors from the system operation loop as much as possible, we propose a new type of runway incursion prevention system, named *anticipatory runway incursion prevention system*, which can predict and detect runway incursions, and then give explicit instructions and/or suggestions to pilots/drivers to avoid the runway incursion/collision. The features of our system include: (1) long-range prediction: to predict runway incursions earlier than current runway incursion prevention systems for reserving more time to handle the incidents, (2) explicit instructions and/or suggestions: not only to give alerts to air traffic controllers/pilots/drivers, but also to give explicit instructions and/or suggestions to pilots/drivers to avoid the incursions/collisions, and (3) flexible customized anticipatory model for different air traffic control policies and airports. To implement our system, we adopted logic-based reasoning [44, 45] for prediction and decision making, and adopted anticipatory reasoning-reacting system [38] as the core of the system architecture. To evaluate our system, we build a simulation system, and evaluate our system using three historical real scenarios of incidents, four conventional test scenarios for evaluating runway incursion prevention systems, as well as some random variation of these scenarios for testing the frequency of missed alerts and false alerts.

6.3 Problems of current runway incursion prevention systems

A number of systems for runway incursion avoidance and detection were proposed, such as runway incursion prevention system (RIPS) [36], airport surface detection equipment - model X (ASDE-X) [68] , runway incursion monitoring and collision avoidance system (RIMCAS) [185], runway status light system (RWSL) [64], airport movement area safety system (AMASS) [61], and advanced surface movement guidance and control system (A-SMGCS) [92, 134].

We investigated and analyzed the problems of current runway incursion prevention systems, then specified what these problems are, and why these problems are important or why these problems arise.

- Tardy prediction/detection

  Although current systems can obtain accurate and sufficient information of aircraft/vehicles, they cannot always predict/detect a runway incursion as early as an experienced air traffic controller predict/detect by using the same surveillance information. There are two reasons arising tardy prediction/detection. First, current systems focus on detecting an ongoing runway incursion, or predicting a forthcoming collision, but not focus on predicting a forthcoming runway incursion. Due to the high speed of aircraft, when a runway incursion has happened, there may be insufficient time to manage the incident [157]. Second, current systems use inflexible algorithms/models, and cannot flexibly utilize the empirical knowledge of air traffic controllers and the concrete conditions of the airport [15]. For example, in the real incident shown in section 6.7, for RIMCAS, only two aircraft simultaneously enter a "critical circle", an alert can be triggered [15]. In contrast, a concentrated experienced air traffic controller could utilize the layout of runways to predict that runway incursion long before the two aircraft enter the "critical circle".

- Lack of explicit instructions and/or suggestions

  Current systems only give alerts, but cannot give explicit instructions and/or suggestions to pilots/drivers to avoid the incursions/collisions. Although air traffic controllers/pilots/drivers may be aware of an ongoing/forthcoming incident, they have to make correct decisions to manage the incidents in very limited spans of time. To make the decisions also consumes time, besides, in an extreme emergency, people tend to relay on instinct instead of rationality.

- High frequency of missed alerts and false alerts

  Several current systems suffer high frequency of missed alerts and false alerts [160]. Missed alerts and false alerts is a direct result of erroneous or missing traffic data [36]. Besides, inflexible algorithms/models may lose their effectiveness for some special situations and airports [15]. Furthermore, numerical calculation approaches cannot distinguish between a severe accident and a trivial incident, which causes high frequency of unnecessary alerts.

- Inflexible algorithms/models

  Most of current system use inflexible algorithms/models to predict/detect incursions/collisions. Inflexible algorithms/models cannot adapt to the needs of customization and variability, i.e., they cannot flexibly utilize the empirical knowledge of air traffic controllers and the concrete conditions of the airports. Besides, airport operations are complex and vary for different policies and different airports (even in different areas of one airport). Therefore, we should not use a single general algorithm/approach for different policies and airports (even in different areas of one airport). Furthermore, as we discussed above, inflexible algorithms/models could cause both tardy prediction/detection and missed/false alerts.

Although some approaches based on artificial intelligence may use flexible models, such as [28], they did not show their effectiveness and efficiency, as well as practical use in real systems.

In this case study, our aim is not to solve all these problems completely. We only focus on (1) earlier prediction/detection of runway incursions/collisions than current systems by using the same processed information of aircraft/vehicles, such as position, speed, and acceleration, (2) facility to give explicit instructions and/or suggestions to avoid the incursions/collisions, and (3) flexible models. It is important for runway incursion prevention systems to get accurate and sufficient surveillance information of aircraft/vehicles, such as position, speed, and acceleration, out of noisy surveillance data in time. Almost all current systems focus on aircraft/vehimissed/false surveillance informationmissed/false surveillance informationcles surveillance, and made big progress on this topic. However, they are beyond the scope of this work that how to get accurate and sufficient surveillance information and how to work under missed/false surveillance information.

6.4 Anticipatory runway incursion prevention systems

ARIPS We propose a new type of runway incursion prevention system, named *anticipatory runway incursion prevention system* (ARIPS), which is a runway incursion prevention system not only predict and detect runway incursions, but also give explicit instructions and/or suggestions to pilots/drivers to avoid runway incursions/collisions. Besides, an APIPS also predicts runway incursions as early as an experienced air traffic controller does, and has low frequency of missed alerts and false alerts and instructions. Due to the problem of inflexible algorithms/models and its consequences, an ARIPS uses flexible models, which can be customized for different air traffic control policies and airports.

We analyzed the system requirements of the ARIPS as follows.

R1 The ARIPS should predict/detect a runway incursion/collision as early as an experienced air traffic controller does.

R2 The ARIPS should give alerts to both air traffic controllers and related pilots/drivers. Besides, the ARIPS should give alerts with different emergency levels, such as advisory, watch, and warning to specify an alert from trivial to severe. System users can choose to turn off trivial alerts to focus on the critical incidents.

R3 Besides alerts, the ARIPS should also give explicit instructions and/or suggestions to pilots/drivers to avoid the incursions/collisions.

R4 The ARIPS should be a general one, which can be deployed in different airports with trivial modification or configuration.

R5 The ARIPS should use flexible models, which can be customized for specific air traffic control policies and airports, as well as different areas of a specific airport. Besides, the models of ARIPS should express the knowledge of air traffic control in an explicit way, thus the domain experts can customize, modify, and examine these models easily.

R6 The ARIPS should have low frequency of missed alerts and false alerts and instructions.

6.5 Applying the methodology

Phase 1: Analyze the target domain

The possible accidents are runway incursions and runway collisions. The possible anticipatory actions include holding a taxiing aircraft/vehicle, stopping taking off, stopping landing and going around, etc.

Phase 2: Analyze the legacy system

The set I includes: 1) the locations of aircrafts/vehicles, and 2) the speed and acceleration of aircrafts/vehicles. These information are provided by *traffic information system* [157] and *traffic surveillance sensor system* [157]. For set O, actions are instructions, which are carried out by *human machine interface system*, which gives the alerts about runway incursions or collisions as well as the instructions for avoiding runway incursions or collisions to both pilots/drivers and air traffic controllers, and *traffic signal system*, which gives instructions to pilots/drivers by mainly using lights through different light systems deployed in the airport. The anticipatory actions are some instructions in O, such as holding a taxiing aircraft/vehicle, stopping taking off, and stopping landing and going around. The legacy system can only give passive warnings, but cannot prevent runway incursions or collisions anticipatorily.

Phase 3: Define requirements of the target system

The target system should not only detect a runway incursion or predict an forthcoming collisions, but also predict a runway incursion as early as an experienced air traffic controller. Besides, the target system should give not only alerts but also explicit instructions and/or suggestions to pilots/drivers to avoid the incursions/collisions.

Phase 4: Construct anticipatory model

We chose *temporal deontic relevant logics* [42] as the logic basis to both predict and make decisions.

For the world model, we built the vocabulary as a predicate dictionary shown in table 6.1. Besides, we also need some empirical knowledge, such as "if a aircraft is accelerating on a runway, then that aircraft is taking off" represented as $\forall o \forall r(At(o,r) \wedge Runway(r) \wedge Aircraft(o) \wedge Accelerate(o) \Rightarrow TakeOffFrom(o,r))$.

In this case, the interesting predictions are runway incursions and runway collisions. The predictive model includes the knowledge used for generating these interesting predictions. For example, we have conditionals $\forall r_1 \forall r_2 \forall r_3 \forall o(C2(r_1,r_2,r_3) \wedge H(At(o,r_1)) \wedge At(o,r_2) \Rightarrow F(At(o,r_3)))$ meaning "if an aircraft has crossed the hold line of a runway, then that aircraft will arrive at that runway", and $F(At(o,r)) \wedge Active(r) \Rightarrow F(RIby(o,r))$ meaning "if an aircraft will arrive at a runway and that runway is active (occupied for taking off or landing), then that aircraft will cause runway incursion on that runway".

The behavioral model specifies the actions to prevent a ongoing/furture runway incursion/collision. For example, we have conditionals $\forall o \forall r(F(At(o,r)) \wedge F(RIby(o,r)) \Rightarrow O(Hold(o)))$ meaning "if an aircraft will arrive at a runway and will cause a runway incursion on that runway, then the aircraft is obligatory to hold in its position", $\forall o_1 \forall o_2 \forall r(F(LandOn(o_1,r)) \wedge F(RIby(o_2,r)) \Rightarrow O(GoAround\ (o_1)))$ meaning "if an aircraft is landing on a runway and there will be a runway incursion on that runway, the aircraft should cancel landing and go around", and $\forall o_1 \forall o_2 \forall r(TakeOffFrom(o_1,r) \wedge F(RIby(o_2,r)) \Rightarrow O(Evade\ (o_1)))$ meaning "if aircraft is taking off form a runway and there will be a runway incursion on that runway, the aircraft should cancel taking off, and evade potential collision".

Phase 5: Implementation

Because we also chose temporal deontic relevant logics as the logic basic. The interesting terms of predictions are $RIby$, $Collision$, $TakeOffFrom$, $LandOn$, $OccupyIntersection$, etc. The interesting terms of decision making are $GoAround$, $Evade$, $Hold$, etc.

Action planner decides an instruction for real time operation. For example, if the speed of aircraft is greater than V1, then that aircraft cannot stop taking off, because V1 is the maximum speed in the takeoff at which the pilot can take the first action (e.g., apply brakes, reduce thrust, deploy speed brakes) to stop the airplane within the accelerate-stop distance.

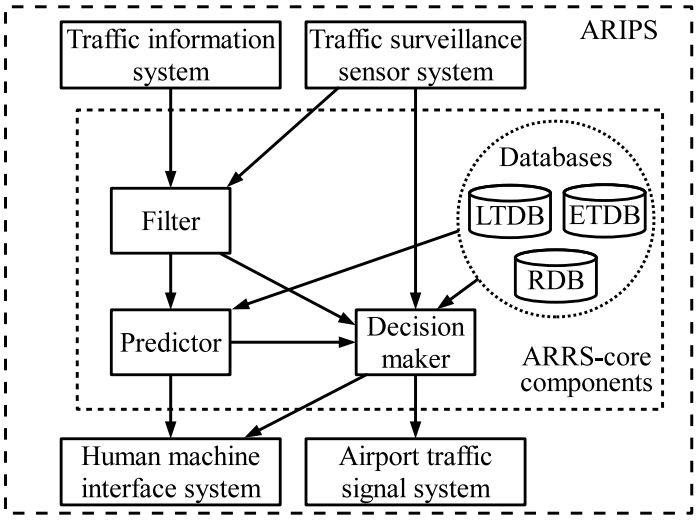

Figure 6.1: An architecture of ARRS-based ARIPS

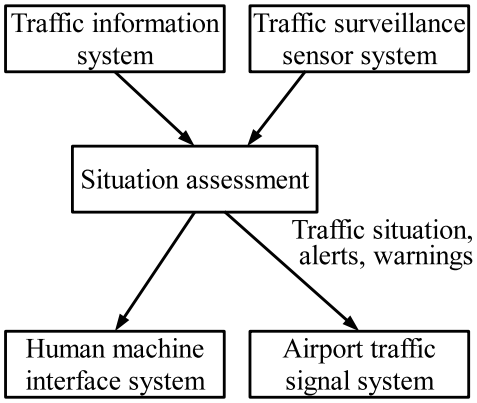

Figure 6.2: A basic architecture of current runway incursion prevention system

6.6 The implemented ARIPS

6.6.1 Overview

An ARIPS contains several components shown in figure 6.1, while in this work, our aim is not to improve current traffic information systems, traffic surveillance sensor systems, human machine interface systems, or airport traffic signal systems, but rather only focus on the core components used for prediction/detection of runway incursions/collisions and decision-making about instructions. To show the difference between current systems and our system, we show a basic architecture of current runway incursion prevention system as figure 6.2, which is according to [157].

6.6.2 System architecture

Figure 6.1 shows an architecture of ARRS-based ARIPS, which includes the following components: traffic information system, traffic surveillance sensor system, filter, predictor, decision maker, databases (LTDB, ETDB, and RDB), human machine interface system, and airport traffic signal system. *Traffic information system* provides local traffic information for the participating users [157]. *Traffic surveillance sensor system* provides information about the environment, which is usually preprocessed into a compact meaningful representation by the sensors [157]. These sensors can be part of an airport infrastructure or part of an aircraft and/or other mobile units [157]. *Filter* filters out the trivial information of aircraft/vehicles and generates the qualitative information of aircraft/vehicles as current situation for predictor and decision maker. *Predictor* receives current situation from the filter, then outputs predictions about runway incursions, collisions, etc. *Decision maker* receives predictions from the predictor, current situation from the filter, and aircraft's/vehicles' real time status from the traffic surveillance sensor system, then outputs instructions for avoiding runway incursions or collisions. *LTDB* is a logical theorem database, which stores fragments of logical systems [38]. *ETDB* is an empirical theory database storing anticipatory model, including world model, predictive model, and behavioral model. *RDB* is a database storing *transform rules*, *calculative rules*, *interesting formula definitions*, and *interesting terms*. *Human machine interface system* gives the alerts about runway incursions or collisions, as well as the instructions for avoiding runway incursions or collisions to both pilots/drivers and air traffic controllers. Besides, air traffic controllers also use the human machine interface system as an input device. *Traffic signal system* gives instructions to pilots/drivers by mainly using lights through different light systems deployed in the airport.

6.6.3 Filtering

Because the ARIPS uses logic-based reasoning method to predict and make decisions, the predictor and decision maker only need qualitative information of aircraft/vehicles. The most useful information for ARIPS is aircraft's/vehicles' locations, as well as their statuses, such as whether an aircraft is accelerating.

The filter consists of a set of filter modules, while each module filters out a certain kind of sensory information. The filter's locating module filters out qualitative location information of aircraft/vehicles, including past locations and current locations. We divide an airport including its surrounding airspace into different regions, and name each region with a unique name, shown in figure 6.3. The locating module represents each region as a polygon, i.e., a set of vertices. Both traffic information system and traffic surveillance sensor system could provide the location of an aircraft/a vehicle by a point, i.e., a pair of coordinate. The locating module calculates which region each aircraft/vehicle locates, by calculating whether a point is inside a given polygon. The locating module maintenances two hash maps: PastLocation<aircraft/vehicle identifier, locating region> and CurrentLocation<aircraft/vehicle identifier, locating region>. When an aircraft/a

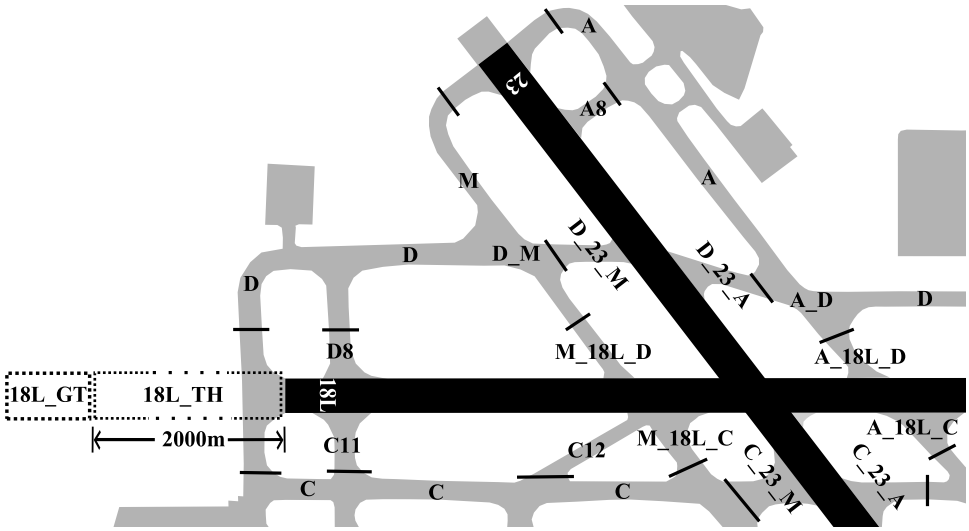

Figure 6.3: Naming different regions of airport with unique names

vehicle move from one region to another one, the locating module use its location stored in CurrentLocation to overwrite its location in PastLocation, then use the new current location to overwrite its location in CurrentLocation. To filter out aircraft's status information, the filter's status module gets acceleration of aircraft from traffic surveillance sensor system, e.g., an aircraft's accelerometer could provide the aircraft's acceleration. If an aircraft's acceleration is greater than a give value, status module calculates that the aircraft is accelerating. The filter modules will not always output filtered information, because sensory date update continuously. Therefore, the filter modules only output filtered information, when any qualitative information has changed. The filter modules output the information as strings, e.g., "N409DR was at A_D", "N409DR is at A_18L_D", "JIA390 is at 18L", and "JIA390 is accelerating".

6.6.4 Predicting

The purpose of prediction of ARIPS is to find out "which aircraft/vehicles will cause runway incursions/collisions?", "which aircraft will be affected by the runway incursions/collisions?", and "where the runway incursions/collisions will happen?"

Figure 6.4 shows the data flow diagram of the predictor. First, formula generator translates the filtered current situation to logical formulas according to transformation rules, which base on the predicate dictionary. Second, the forward reasoning engine gets input of these logical formulas about current state, the world model, the predictive model, and the fragment of logical system [38], and then applies forward reasoning. Third, the formula chooser chooses predictions according to the interesting formula definitions and interesting terms from all formulas reasoned out by forward reasoning engine, and transfers them to decision maker and human machine interface system.

Because the ARIPS uses logic-based reasoning method to predict and make de-

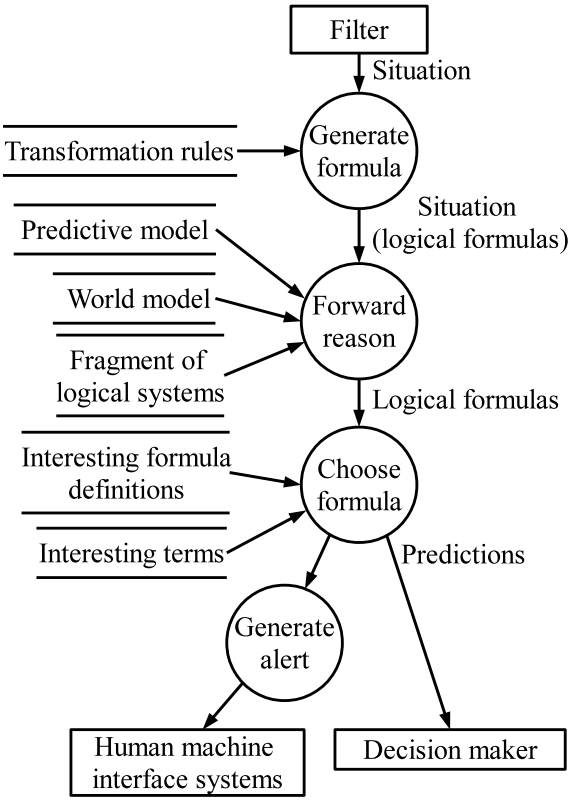

Figure 6.4: Predictor's data flow diagram

cisions, we need to choose a logical basis to underlie temporal reasoning and deontic reasoning. Therefore, we chose *temporal deontic relevant logics, cheng2006temporal* as the logic basis to both predict and make decisions.

A *world model* is a set of empirical theories represented by logical formulas in the target domain except empirical theories related with time and behavior [71]. The world model has two functions: to represent the static conditions of the airport and to represent essential empirical knowledge except related with time and behavior. A *predicate vocabulary* specifies a vocabulary to represent the status of the real world, shown in table 6.1. By using the predicate vocabulary, we can represent the airport diagram as a set of logical formulas, and represent the current situation as a set of logical formulas. We use figure 6.3 as an example to show how to represent the airport. First, we divide an airport including its surrounding airspace into different regions, and name each region with a unique name, i.e., a constant symbol, shown in figure 6.3. Second, we use a formula constructed by a predicate and the constant symbol to represent what is that region, e.g., $Runway(18L)$ representing region $18L$ is a runway, $Way(A\_18L\_D)$ representing that it is a section of way, and A_D representing that it is an intersection. Third, we use a formula constructed by a predicate and constant symbols to represent the relationship of different regions, e.g., $C(18L, A\_18L\_D)$ representing region $18L$ and region A_18L_D connect with each other, and $C2(A\_D, A\_18L\_D, 18L)$ representing that region $18L$ is a runway, A_18L_D is a section of way, region

A_D connects with region A_18L_D, and region A_18L_D connects with region $18L$. These formulas represent the topology construction of the airport, which include all geographic information that the system needs. To transform the situation information from the filter into logical formulas is also based on the predicate dictionary. Figure 6.5 shows how to transform situation information represented by strings into logical formulas. "P" is a temporal operator, while PA meaning "it has been the case at least once in the past up to now that A". Next step is to determine the common knowledge involving in the target problem and to represent the knowledge as conditionals. Specifically, the knowledge in world model does not include knowledge related with prediction and behavior. For example, "if an aircraft is accelerating on a runway, then that aircraft is taking off" can be written as: $\forall o \forall r(At(o,r) \wedge Runway(r) \wedge Aircraft(o) \wedge Accelerate(o) \Rightarrow TakeOffFrom(o,r))$ (**WM1**). "If an aircraft is taking off from a runway, then that runway is active" is written as $\forall o \forall r(TakeOffFrom(o,r) \Rightarrow Active(r))$ (**WM2**). "If an aircraft will land on a runway, then that runway is active" is written as $\forall o \forall r(F(LandOn(o,r)) \Rightarrow Active(r))$ (**WM3**). "F" is a temporal operator, while FA meaning "it will be the case at least once in the future form now that A". "If an aircraft is taking off from a runway besides that runway intersects with another runway, then that aircraft is occupying the two intersecting runways" is written as $\forall o \forall r_1 \forall r_2(TakeOffFrom(o,r_1) \wedge Intersecting(r_1,r_2) \Rightarrow OccupyIntersection(o,r_1,r_2))$ (**WM4**).

A *predictive model* is a set of empirical theories, which are represented by logical formulas and related with time in a target domain of the system [71]. The predictive model represents the predictive knowledge used to make prediction. Predictive knowledge is time-related. A piece of predictive knowledge specifies, that when a certain state (both past and current) occurred, some following state will be true in the future. In ARRSs, predictive knowledge is a set of conditionals whose consequent of the future is true if and only if the antecedent holds. Because for any prediction, both the predicted thing and its truth must be unknown before the completion of that prediction, the conclusion should not include the knowledge what we have known. Besides, the premises and conclusion should be relevant. Furthermore, we are only interested in certain kinds of predictions, while in ARIPS, they are mainly collisions, runway incursions, and other predictions, which can help to predict collisions and runway incursions. Here are some examples: $\forall o \forall r_1 \forall r_2 \forall r_3(C2(r_1,r_2,r_3) \wedge H(At(o,r_1)) \wedge At(o,r_2) \Rightarrow F(At(o,r_3)))$ (**PM1**), $\forall o \forall r(F(At(o,r)) \wedge Active(r) \Rightarrow F(RIby(o,r)))$ (**PM2**), $\forall o \forall r_1 \forall r_2 \forall r_3(C3(r_1,r_2,r_3) \wedge H(At(o,r_1)) \wedge At(o,r_2) \Rightarrow F(LandOn(o,r_3)))$ (**PM3**), $\forall o_1 \forall o_2 \forall r(TakeOffFrom(o_1,r) \wedge F(LandOn(o_2,r)) \Rightarrow F(RIby(o_2,r)))$ (**PM4**), $\forall o_1 \forall o_2 \forall r_1 \forall r_2(OccupiedIntersecting(o_1,r_1,r_2) \wedge TakeOffFrom\,(o_2,r_2) \Rightarrow F(RIby\,(o_2,r_1)))$ (**PM5**), $\forall o_1 \forall o_2 \forall r(TakeOffFrom(o_1,r) \wedge TakeOffFrom(o_2,r) \Rightarrow F\,(Collision(o_1,o_2)))$ (**PM6**). **PM1** means when an aircraft has crossed the hold line of a runway, that aircraft will arrive at that runway. **PM1** is valid because aircraft cannot draw back, when an aircraft reach one end of a way, the aircraft will arrive at the other end. Because the above formulas are easy to understand, we do not explain them one by one.

The forward reasoning is performed by a *forward reasoning engine*, which is a

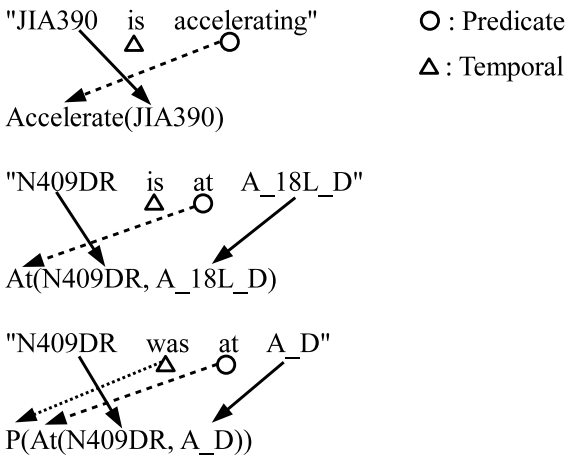

Figure 6.5: Transforming sensory information into logical formulas

program to automatically draw new conclusions by repeatedly applying inference rules to given premises and obtained conclusions until some previously specified conditions are satisfied [47]. In ARIPS, we chose FreeEnCal [47] as the forward reasoning engine.

The formula chooser chooses predictions from all formulas reasoned out by forward reasoning engine, and transfers them to decision maker and other components such as human machine interface systems. We call predictions *interesting formulas* (IF), which is defined as follows [159]: If A is an interesting term, then A is an IF; If A is an interesting term, then $\neg A$ is an IF; If A or B is IFs, then $U(A, B)$ is an IF; If A is an IF, then $\forall xA$ and $\exists xA$ are IFs; If A is an IF, then ΦA is an IF, where Φ is one of $\{G,\ F\}$. The interesting terms are $RIby$, $Collision$, $TakeOffFrom$, $LandOn$, $OccupyIntersection$, etc.

Alert generator gets the prediction and generates alerts with different emergency levels, such as $RIby$ and $F(Collision)$ having highest emergency level, and $F(RIby)$ taking second place.

6.6.5 Decision making

There are two phases to choose instructions for avoiding the runway incursions/collisions: first phase is to make qualitative decision by logic-based reasoning, and second phase is to refine the decision by quantitative calculation. Figure 6.6 shows the data flow diagram of the decision maker.

Qualitative decision: The progress of qualitative decision making by logic-based reasoning is similar with prediction by logic-based reasoning, where the main difference is to use behavioral model instead of predictive model. The purpose of qualitative decision is to find out "which actions must be taken?" and "which actions should be taken?" Thus, the result of qualitative decision is a set of candidates of next actions, which are labeled as "obligatory"/"permitted" and/or priorities. A *behavioral model* is a set of empirical theories that are represented by logical formulas and related with behavior in a target domain of the system [71]. The

Table 6.1: Predicate vocabulary

Formula	Meaning
$Aircraft(o)$	o is an aircraft.
$Vehicle(o)$	o is a vehicle.
$Other(o)$	o is an object cannot communicate with air traffic controllers.
$Runway(r)$	r is a runway.
$Taxiway(r)$	r is a taxiway.
$Intersection(r)$	r is an intersection.
$Way(r)$	r is a section of way.
$Approach(r)$	r is an area region on extension line of a runway, from the runway threshold to 2000 meters far away from the runway threshold.
$HotSpot(r)$	r is a hot spot, which is defined as a location on an airport movement area with a history of potential risk of collision or runway incursion.
$Active(r)$	Runway r is occupied for taking off or landing.
$Accelerate(o)$	o is notability accelerating.
$At(o, r)$	o is in region r.
$RIby(o, r)$	o is causing a runway incursion on runway r.
$Collision(o_1, o_2)$	o_1 and o_2 collide on the runway.
$TakeOffFrom(o, r)$	Aircraft o is taking off from runway r.
$LandOn(o, r)$	Aircraft o is landing on runway r.
$OccupyIntersection$ (o, r_1, r_2)	Aircraft o occupies two intersecting runways. r_1 and r_2 for taking off or landing.
$Intersecting(r_1, r_2)$	Runway r_1 and r_2 cross each other.
$C(r_1, r_2)$	region r_1 connects to region r_2.
$C2(r_1, r_2, r_3)$	$Runway(r_3) \wedge C(r_1, r_2) \wedge C(r_2, r_3) \wedge Way(r_2)$
$C3(r_1, r_2, r_3)$	$Runway(r_3) \wedge Approach(r_2) \wedge$ $C(r_1, r_2) \wedge C(r_2, r_3)$

behavioral model specifies that, when a certain event or state occurs, which actions must/should be taken, as well as when a prediction about certain event or state is made, which anticipatory actions must/should be taken. The behavioral model is assembled by conditionals, which can be used to get these results of qualitative decision. Here are some examples: $\forall o \forall r(F(LandOn(o, r)) \wedge F(RIby(o, r)) \Rightarrow O(GoAround(o)))$ (**BM1**), $\forall o_1 \forall o_2 \forall r(TakeOffFrom(o_1, r) \wedge F(RIby(o_2, r)) \Rightarrow O(Evade(o_1)))$ (**BM2**), $\forall o \forall r(F(At(o, r)) \wedge F(RIby(o, r)) \Rightarrow O(Hold(o)))$ (**BM3**), $\forall o_1 \forall o_2 \forall r(F(LandOn(o_1, r)) \wedge F(RIby(o_2, r)) \Rightarrow O(GoAround(o_1)))$ (**BM4**), $\forall o \forall r_1 \forall r_2$ $(TakeOffFrom(o, r_1) \wedge F(RIby(o, r_2)) \Rightarrow O(Evade(o)))$ (**BM5**). "O" is a deontic operator, while OA meaning "it is obligatory that A". "$GoAround(o)$" is a go-around instruction that an aborted landing of an aircraft o that is on final approach. "$Hold(o)$" is a hold-in-position instruction that a taxiing aircraft/vehicle stops going forward, stays in the current position. "Evade(o)" is an instruction

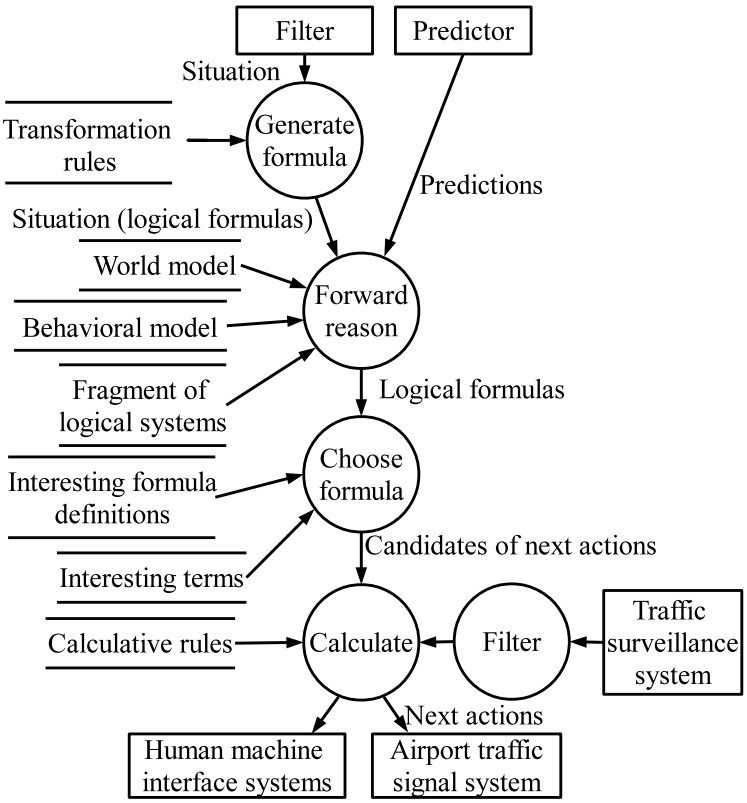

Figure 6.6: Decision maker's data flow diagram

that an aborted taking off of an aircraft o as well as "see and void" a potential collision. To choose candidates of actions, the *interesting formulas* (IF) is defined as follows: If A is an interesting term, then A is an IF; If A is an interesting term, then $\neg A$ is an IF; If A is an IF, then $\forall x A$ and $\exists x A$ are IFs; If A is an IF, then ΦA is an IF, where Φ is one of $\{O, P\}$. The interesting terms are $GoAround$, $Evade$, $Hold$, etc.

Quantitative calculation: In order to decide an instruction for real time operation, quantitative calculation is necessary. For example, if the speed of aircraft o is greater than V1, then aircraft o cannot execute "Evade(o)", because V1 is the maximum speed in the takeoff at which the pilot can take the first action (e.g., apply brakes, reduce thrust, deploy speed brakes) to stop the airplane within the accelerate-stop distance.

6.6.6 Databases

In our current implementation of ARIPS, the databases LTDB, ETDB, and RDB just store the corresponding data for initializing predictor and decision maker, while these databases do not participate in the phase of prediction/detection and decision-making. In the initial phase of ARIPS, predictor reads transformation rules, interesting formula definitions, and interesting terms from RDB, reads fragment of logical system from LTDB, and reads world model and predictive model

from ETDB, while decision maker reads transformation rules, interesting formula definitions, interesting terms, and calculative rules from RDB, reads fragment of logical system from LTDB, ands read world model and behavioral model from ETDB. Both predictor and decision maker stores these data in their working space for the sake of efficiency. When any data in RDB, LTDB, and ETDB changes in run time, predictor and decision maker will updates their corresponding data.

6.6.7 Ad hoc methods for efficiency

Traditionally, logic-based reasoning may be not so efficient and do not satisfy the requirement of time restriction for runway incursion prevention. Especially, the current FreeEnCal does not handle high degree of logic connectives/operators [47] efficiently.

To improve the performance of ARIPS, we adopted some ad hoc methods. First, the formula generator generates the conjunctions of premises, whose degree of "$\wedge$" greater than or equal to 3, then inputs these formulas as redundancy formulas into FreeEnCal. Second, we try to ensure the degree of "$\wedge$" of empirical conditionals in anticipatory model is less than 3. Third, we use multi-processing/multi-threading of FreeEnCal in both predictor and decision maker. When the situations of aircraft/vehicles change, predictor/decision maker creates a new process/thread of FreeEnCal for reasoning. Therefore, there may be several forward reasoning process/thread running simultaneously in predictor or decision maker. Multi-process/thread of FreeEnCal may deduce redundant results. However, ARIPS just output these redundant predictions/instructions one by one, which do not affect the correctness of alerts and instructions.

6.7 System mechanism

In order to explain the mechanism of ARIPS, we use a real incident scenario to show the working process of ARIPS. The incident occurred on 18 June 2010 at Zurich airport [15], shown in figure 6.7. At 12:00:30 (UTC) the aircraft THA971 received clearance to taxi to the take-off position on runway 16. At 12:01:31 the aircraft BCI937 received clearance to taxi to the take-off position on runway 28. The aircraft BAW713 was ready to depart at holding position point B to the north of the threshold of runway 28. At 12:02:26 the crew of THA971 received clearance to take off from runway 16; they acknowledged it immediately and initiated the take-off roll. At 12:02:31 the crew of BCI937 initiated their take-off roll on runway 28 (Ideally, air traffic controller could notice the runway incursion at this moment). At 12:02:47, the crew of BAW713 informed the air traffic controller that at that moment it was possible that two aircraft would take off simultaneously. At 12:02:50 the air traffic controller instructed the crew of BCI937 to abort take-off. The crew obeyed this instruction and vacated runway 28 on taxiway A4. The crew of THA971 continued their take-off and flight to their destination. At 12:03:01, the RIMCAS generated a stage 2 alert.

How ARIPS works under this scenario is as follows. In the initial phase

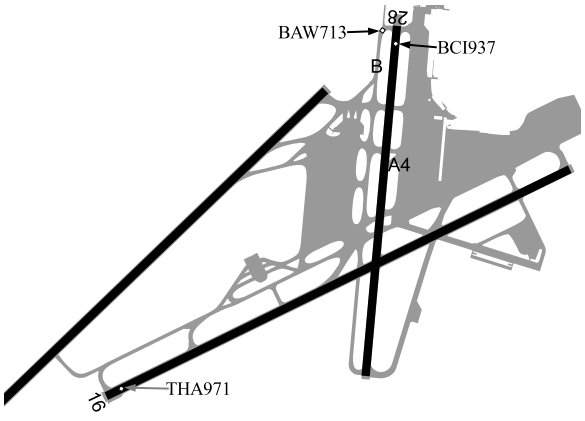

Figure 6.7: An incident occurred on 18 June 2010 at Zurich airport

of ARIPS, predictor and decision maker read the anticipatory model and other data from LTDB, ETDB, and RDB, as shown in section 6.6.6. The anticipatory model contained all empirical theorems shown in section 6.6. Besides, the world model also contained static conditions of the airport, such as $Runway(28)$, $Runway(16)$, and $Intersecting(28, 16)$. After initialization, the filter continuously received each aircraft's location, speed, and acceleration from traffic information system and traffic surveillance sensor system, and filtered out the qualitative information of aircraft for predictor and decision maker. At 12:02:31, the filter filtered out $Aircraft(THA971)$, $Aircraft(BCI937)$, $At(THA971, 16)$, $At(BCI937, 28)$, $Accelerate(THA971)$, and $Accelerate(BCI937)$. Once filter updated information, predictor used this information for prediction, while decision maker used this information for decision-making. From (1) above facts at 12:02:31, (2) above static conditions of the airport from world model, (3) empirical theorems $\forall o \forall r(At(o, r) \wedge Runway(r) \wedge Aircraft(o) \wedge Accelerate(o) \Rightarrow TakeOffFrom(o, r))$ (**WM1**) and $\forall o \forall r_1 \forall r_2(TakeOffFrom(o, r_1) \wedge Intersecting(r_1, r_2) \Rightarrow OccupyIntersection(o, r_1, r_2))$ (**WM4**) of world model, and (4) empirical theorem $\forall o_1 \forall o_2 \forall r_1 \forall r_2(OccupiedIntersecting(o_1, r_1, r_2) \wedge TakeOffFrom(o_2, r_2) \Rightarrow F(RIby(o_2, r_1)))$ (**PM5**) of predictive model, based on temporal deontic relevant logics, the predictor deduced predictions/detections $TakeOffFrom(BCI937, 28)$, $TakeOffFrom(THA971, 16)$, and $F(RIby(BCI937, 16))$, then sent these predictions/detections to decision maker and human machine interface system. Then human machine interface system gave alert $F(RIby(BCI937, 16))$, i.e., "BCI937 will cause a runway incursion on runway 16", to both air traffic controller and pilot of BCI937. From the above predictions/detections and empirical theorems $\forall o_1 \forall o_2 \forall r(TakeOffFrom(o_1, r) \wedge F(RIby(o_2, r)) \Rightarrow O(Evade(o_1)))$ (**BM2**) and $\forall o \forall r_1 \forall r_2(TakeOffFrom(o, r_1) \wedge F(RIby(o, r_2)) \Rightarrow O(Evade(o)))$ (**BM5**), based on temporal deontic relevant logics, the decision maker deduced instructions $O(Evade(BCI937))$ and $O(Evade(THA971))$. Because the speed of either BCI937 or THA971 was less than V1, according to the calculative rule, the final instructions were "BCI937 abort take-off" and "THA971 abort take-off", while decision maker gave these instructions to both human machine interface system and airport traffic signal system. Human machine interface system gave these

instructions to air traffic controller and pilots of BCI937 and THA971, while airport traffic signal system also gave signal instruction to both BCI937 and THA971 for aborting take-off.

We compare ARIPS with the RIMCAS deployed in Zurich airport. In the real incident, when potential runway incursion occurred 30 seconds, the RIMCAS generated a stage 2 alert. In contrast, in our simulation experiments shown in section 6.8, the ARIPS only use 7 seconds to give alert of prediction of runway incursion as well as the instructions to prevent the runway incursion.

6.8 Simulation experiments

The purpose of simulation experiments is to show the performance, correctness, and generality of ARIPS. Because the difference between current RIPSs and ARIPS is the core components of ARIPS, we only implemented and tested the core components of ARIPS.

To prepare input data and to simulate outside of the core components of ARIPS, we also built a simulation program to simulate real dynamic airports. The simulation program has following features: (1) to simulate any existing airports or user-define airports, (2) user-define aircraft/vehicles routes and speeds (supporting speed change), (3) tools for user-define airports and aircraft/vehicles routes, (4) aware of collision and display that, (5) interface for ARIPS getting information of aircraft/vehicles, including location, speed, and acceleration, and (6) graphic interface to display the dynamic airport (aircraft taxiing, aircraft taking off and landing), which is similar with ASDE-X. All programs ran on a PC with Intel Core i7-860 Processor (2.8GHz, 4 cores, 8 threads), 4Gbyte memory, and Scientific Linux release 6.2 for x86_64 (Linux kernel is 2.6.32-220). All programs are written in Java and running on OpenJDK Runtime Environment (IcedTea6 1.10.4), except FreeEnCal in C++. Because the purpose of this work is not about how to get surveillance information of aircraft/vehicles, we used real surveillance information about the position, speed and acceleration of aircraft from the three historical incidents and four test scenarios [36] as the basic experimental data instead of raw sensory data. Besides, we also prepared variations of above scenarios as experimental data.

To evaluate the performance of ARIPS, we compared the execution time of ARIPS with current RIPSs and/or human using the same surveillance information from the three historical incidents and four test scenarios. The execution time of current RIPSs and/or human is obtained directly from related documents. Figure 6.8 shows the meaning of execution time. For human, the total time of prediction/detection is time from the incident occurs to the ATC controller considers the incident or gives the alert about the incident, and the total time of decision-making for instructions is from the incident occurs to the ATC controller gives the ATC-commands for handling the incident. For current RIPSs, the total time of prediction/detection is time from the incident occurs to the RIPS gives alerts about the incidents. For ARIPS, the total time of prediction/detection is the sum of the lead time for updating surveillance information and the time for predict-

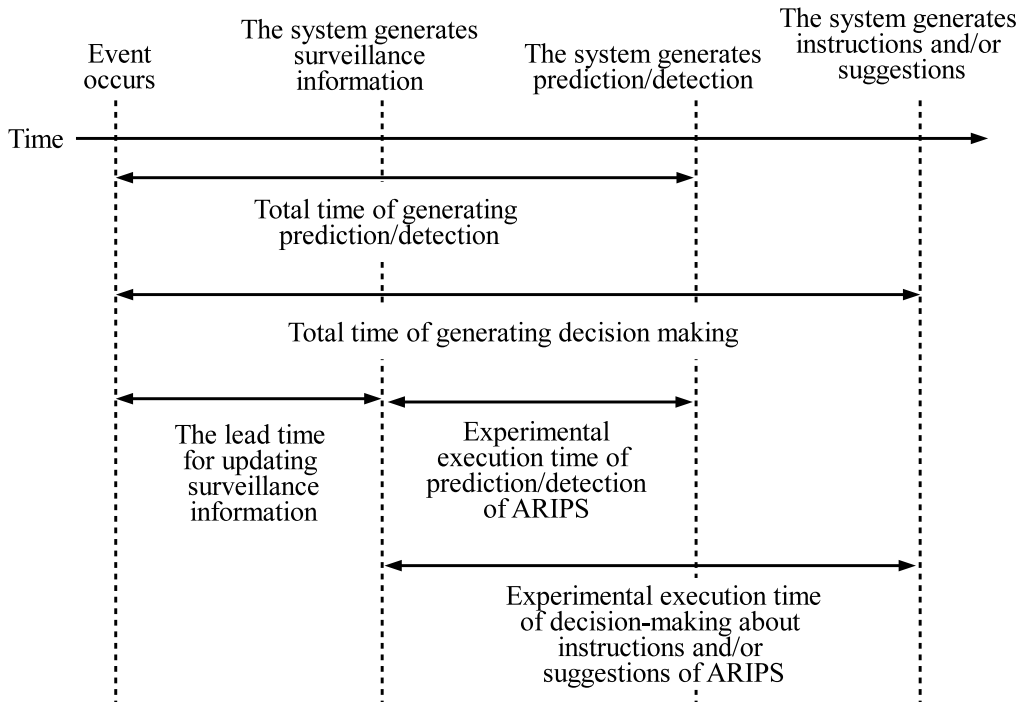

Figure 6.8: Meaning of execution time

Table 6.2: Airport surface movement surveillance performance.
Lines beginning with req. show the requirements for(ASDEX/ASMGCS).
Lines beginning with emp. show the performance experienced in field studies.

Indicator	ADS-B	SMR(i)	MLAT
Req. target report update rate	1Hz	1Hz	2Hz
Emp. target report update rate	1Hz/0.1Hz	1Hz	2Hz
Req. delay	0.25s	0.25s	0.25s
Emp. delay	0.25s/2.0s	0.25s	0.25s/0.5s

ing/detecting the incursion/collision using the surveillance information, and the total time of decision-making for instructions and/or suggestions is the sum of the lead time for updating surveillance information, the time for prediction/detection, and the time for generating the decision. In the experiments, we did not simulate the process of generating surveillance information, but used surveillance information as input data of ARIPS directly. To compensate the time for updating surveillance information, we introduced the lead time of updating surveillance information, and added the lead time to the execution time of ARIPS additionally. Table 6.2 shows the surveillance performance of some traffic surveillance sensor systems [157]. According to this table, the experimental lead times of ADS-B, SMR(i), and MLAT are 0.25 - 2 seconds, 0.25 - 1 seconds, 0.25 - 0.5 seconds correspondingly. In the experiments, we adopted worst lead time, 2 seconds.

A summary of experimental results are as follows. Because we did each scenario

experiment at least five times, our result showed a range of execution time of ARIPS.

- *Scenario A*: Boston Logan International Airport (BOS), Nov. 24, 2010 [133]

  Type: Departure/Taxi

  In the real incident: The ATC controller used 9 seconds to react, and 2 more seconds to give instructions. The runway incursion prevention system did not work.

  ARIPS's performance: Using 3.4 - 4.0 seconds to give alert "$F(RIby(JB1264, 15R))$" and instructions "$O(Evade(JBU417))$" and "$O(Hold(JBU1264))$".

  T1 = T3 + T4 = 2.0 + 4.0 = 6.0

  T2 = T3 + T5 = 2.0 + 4.0 = 6.0

- *Scenario B*: Charlotte Douglas International Airport (CLT), May 29, 2009 [62]

  Type: Departure/Taxi

  In the real incident: After the incident occurred, 14 seconds later, ASDE-X alert was given.

  ARIPS's performance: Using 2.6 - 3.8 seconds to give alert "$F(RIby(N409DR, 18L))$" and instructions "$O(Evade(JIA390))$" and "$O(Hold(N409DR))$".

  T1 = T3 + T4 = 2.0 + 3.8 = 5.8

  T2 = T3 + T5 = 2.0 + 3.8 = 5.8

- *Scenario C*: Zurich Airport (LSZH), June 18, 2010 [15] - the incident shown in section 6.7.

  Type: Departure/Departure on intersecting runways

  In the real incident: After the incident occurred, 16 seconds later, the crew of BWA713 reported the runway incursion to ATC Controllers. After the incident occurred, 19 seconds later, ATC controller instructed BCI937 to abort take-off. After the incident occurred, 30 seconds later, the RIMCAS system generates a stage 2 alert.

  ARIPS's performance: Using 4.9 - 5.0 seconds to give alert "$F(RIby(BCI937, 18L))$" and instructions "$O(Evade(BCI937))$" and "$O(Evade(THA971))$".

  T1 = T3 + T4 = 2.0 + 5.0 = 7.0

  T2 = T3 + T5 = 2.0 + 5.0 = 7.0

- *Scenario D*: Test scenario 1 on Dallas-Fort Worth International Airport (DFW), NASA, 2002 [36]

  Type: Arrival/Taxi

  Performance of NASA's Runway incursion prevention system (RIPS): After the incident occurred, 7 seconds later, runway incursion advisory and alerting system (RIAAS) gave runway traffic alert.

ARIPS's performance: Using 4.4 - 4.7 seconds to give alert "$F(RIby(Taxi, 35L))$" and instructions "$O(Hold(Taxi))$" and "$O(Evade(Arrival))$".

T1 = T3 + T4 = 2.0 + 4.7 = 6.7

T2 = T3 + T5 = 2.0 + 4.7 = 6.7

- *Scenario E*: Test scenario 2 on DFW, NASA, 2002 [36]

  Type: Departure/Taxi

  Performance of RIPS: After the incident occurred, 15 seconds later, RIAAS gave runway traffic alert.

  ARIPS's performance: Using 4.4 - 4.7 seconds to give alert "$F(RIby(Taxi, 35L))$" and instructions "$O(Evade(Departure))$" and "$O(Hold(Taxi))$".

  T1 = T3 + T4 = 2.0 + 4.7 = 6.7

  T2 = T3 + T5 = 2.0 + 4.7 = 6.7

- *Scenario F*: Test scenario 3 on DFW, NASA, 2002 [36]

  Type: Taxi/Departure

  Performance of RIPS: After the incident occurred, 10 seconds later, RIAAS gave runway traffic alert.

  ARIPS's performance: Using 4.1 - 4.5 seconds to give alert "$F(RIby(Taxi, 35L))$" and instructions "$O(Evade(Departure))$" and "$O(Hold(Taxi))$".

  T1 = T3 + T4 = 2.0 + 4.5 = 6.5

  T2 = T3 + T5 = 2.0 + 4.5 = 6.5

- *Scenario G*: Test scenario 4 on DFW, NASA, 2002 [36]

  Type: Arrival/Departure

  Performance of RIPS: After the incident occurred, 13 seconds later, RIAAS gave runway traffic alert.

  ARIPS's performance: Using 2.5 - 2.6 seconds to give alert "$F(RIby(Arrival, 35L))$", while using 4.4 - 4.9 seconds to give instructions "$O(GoAround(Arrival))$" and "$O(Evade(Departure))$".

  T1 = T3 + T4 = 2.0 + 2.6 = 4.6

  T2 = T3 + T5 = 2.0 + 4.9 = 6.9

Table 6.3 compares current RIPS and/or human with ARIPS in the total time of prediction/detection and total time of decision making for instructions. We adopted worst lead time of current traffic surveillance sensor systems (2 seconds) and worst execution time of ARIPS. We could conclude that ARIPS provided earlier prediction of incidents and earlier decision-making for instructions than current RIPSs and/or humans in all scenarios.

The following contents explain the reason why ARIPS could predict earlier than conventional RIPSs in the scenarios of three real incidents based on the same sensory data. Conventional RIPSs use inflexible algorithms/models, thus they cannot

Table 6.3: Comparing current RIPS and/or human with ARIPS in the total time of prediction/detection and total time of decision making for instructions

Scenario	System/Human	Total time of prediction/ detection (T1) (sec)	Total time of decision-making (T2) (sec)
A	ATC controller	9	11
	Current RIPS	did not work	N/A
	ARIPS	6.0	6.0
B	Current RIPS	14	N/A
	ARIPS	5.8	5.8
C	ATC controller*	16	19
	Current RIPS	30	N/A
	ARIPS	7.0	7.0
D	Current RIPS	7	N/A
	ARIPS	6.7	6.7
E	Current RIPS	15	N/A
	ARIPS	6.7	6.7
F	Current RIPS	10	N/A
	ARIPS	6.5	6.5
G	Current RIPS	13	N/A
	ARIPS	4.6	6.9

* with the help of crew of aircraft

flexibly utilize the empirical knowledge of air traffic controllers and the concrete condition of a certain airport or certain area of that airport. For example, in the real incident occurred on LSZH, after potential runway incursion occurred, 30 seconds later, the RIMCAS generated a stage 2 alert, because "every second, the speed and directional vector are determined from the current position by calculation. In the process, the directional vector is continuously projected forward. The speed must be higher than 12 meters per second. In order to recognize the problem of two aircraft crossing on two different runways, a circular area with a diameter of 400 meters was laid around the intersection of runways 16/28. If, on the basis of the calculated projections, two aircraft simultaneously enter this 'critical circle' a Stage 2 alert is triggered." [15] In contrast, ARIPS use flexible models, which include the empirical knowledge about the runway intersection such as **WM4** and **PM5**, and the fact that runway 28 and runway 16 cross each other. Therefore, ARIPS could work like real air traffic controllers more than conventional RIPSs, thus ARIPS could provide earlier alerts.

We evaluated the correctness of predictions/detections and decisions about instructions and/or suggestions of ARIPS using scenario A - G, five variations for each test scenarios, and combination of test scenarios of NASA, i.e., three aircraft/vehicles involved in a runway incursion simultaneously. The result of

prediction/detection and decision about instructions and/or suggestions of ARIPS for scenario A - G have been given in the results of performance experiments. In all experiments, predictions/detections are correct, and all instructions are effective to avoid the incursion/collisions.

We evaluated the generality of ARIPS using different scenarios, different airport, and flexible control policy. Scenario A - G and five variations for each test scenarios provides the empirical data about different airports and different incident scenarios. For flexible control policy, we designed following experiment. We assumed air traffic controllers of DFW decided to use the parallel runway 35L for landing and runway 35C for take-off. Therefore, we added $\forall o \forall r(TakeOffFrom(o,r) \wedge LandingRunway(r) \Rightarrow RunwayConfusion(o))$, $\forall o \forall r$ $(TakeOffRunway(r) \wedge LandOn(o,r) \Rightarrow RunwayConfusion(o))$, $LandingRunway(35L)$, and $TakeOffRunway(35C)$ in world model, added $\forall o \forall r(RunwayConfusion(o) \wedge$ $TakeOffFrom(o,r) \Rightarrow P(AbortTakeOff(o)))$ and $\forall o \forall r(RunwayConfusion(o) \wedge$ $F(LandOn(o,r)) \Rightarrow P(GoAround(o)))$ ("P" is a deontic operator, while PA means "it is permitted that A") in behavioral model, added $RunwayConfusion$ as interesting term of predictor, and added $AbortTakeOff$ as interesting term of decision maker. *Scenario*: The departure aircraft was taking off from runway 35L. *ARIPS's performance*: Using 4.6 - 4.8 seconds to give alert "$RunwayConfusion$ $(Departure)$", and instructions "$P(AbortTakeOff(Departure))$". T1 = T3 + T4 = 2.0 + 4.8 = 6.8. T2 = T3 + T5 = 2.0 + 4.8 = 6.8.

In our experiments, scenario A - G and five variations for each test scenarios showed that our system could deal with different airports and different incident scenarios based on flexible models. The experiment about flexible control policy showed that our system could change control policy on run time by changing behavioral model.

Simulation experiments showed that: (1) our system provided earlier prediction of incidents than current systems in the experimental scenarios, (2) our system could give explicit instructions and/or suggestions for handling the incidents, and (3) our system uses flexible model, which can be customized for specific air traffic control policies and airports. However, our current implementation cannot solve the problems of tardy, false or missed alerts caused by erroneous or missing sensory data.

6.9 Discussion

In our experiments, four test scenarios combined with three incident scenarios showed that our system could deal with different airports and different incident scenarios based on flexible models. In additional, the last experiment showed that our system could change control policy on run time by changing behavioral model.

The following contents explain the reason why ARIPS could predict earlier than conventional RIPSs in the scenarios of three real incidents based on the same sensory data. Conventional RIPSs use inflexible algorithms/models, thus they cannot flexibly utilize the empirical knowledge of air traffic controllers and the concrete condition of a certain airport or certain area of that airport. For exam-

ple, in the real incident happened on LSZH in section 6.8, when potential runway incursion happened 30 seconds, the RIMCAS generated a stage 2 alert, because "every second, the speed and directional vector are determined from the current position by calculation. In the process, the directional vector is continuously projected forward. The speed must be higher than 12 meters per second. In order to recognize the problem of two aircraft crossing on two different runways, a circular area with a diameter of 400 meters was laid around the intersection of runways 16/28. If, on the basis of the calculated projections, two aircraft simultaneously enter this 'critical circle' a Stage 2 alert is triggered. [15]" In contrast, ARIPS use flexible models, which include the empirical knowledge about the runway intersection such as **WM4** and **PM5**, and the fact that runway 28 and runway 16 cross each other. Therefore, ARIPS could work like real air traffic controllers more than conventional RIPSs, thus ARIPS could provide earlier alerts.

Simulation experiments showed that: (1) our system uses flexible model, which can be customized for specific air traffic control policies and airports, (2) our system provided earlier prediction of incidents than current systems in the scenarios of three real incidents, and (3) our system could give explicit instructions and/or suggestions for handling the incidents. However, our current implementation cannot solve the problems of tardy, false or missed alerts caused by erroneous or missing sensory data.

6.10 Comparison with related work

There are some applications using artificial intelligence for runway safety, such as [28]. The logic-based reasoning approach for prediction and decision-making used in ARRSs is different from other methods in the following points. First, the logic-based reasoning approach used in ARRSs is to explicitly separate the underlying logical system, reasoning/computing mechanism, and empirical knowledge in any prediction/decision making. Second, the underlying logical system used in ARRSs belongs to the family of strong relevant (relevance) logics [41]. Detailed explain of the differences refer to [44, 45].

Kitajima et al. proposed a decision maker in an ARRS for terminal radar control [105]. The differences between our work and the work of Kitajima et al. are listed as follows. (1) Runway incursion and terminal radar control are totally two different problems. (2) Kitajima et al. only presented a decision maker, not a whole system. (3) Kitajima et al. did not show a convincing evaluation. First, their experiment data is unfounded. Second, the experiment results only include the execution time and number of candidate, without the effectiveness of the chose actions.

Besides ARRS for terminal radar control, there are also other ARRSs. The most important feature of logic-based reasoning approach used in ARRSs is generality. That means we can apply logic-based reasoning approach to any areas for predicting or decision making, by changing the logical system and empirical knowledge. In ARRSs, the empirical knowledge is the anticipatory model. Therefore, in terms of logic-based reasoning approach, the major differences of different

ARRSs are different logical systems and/or anticipatory models.

6.11 Summary

We analyzed the problems of current runway incursion systems, and proposed a new type of runway incursion prevention system based on logic-based reasoning, which can predict and detect runway incursions, then give explicit instructions and/or suggestions to pilots/drivers to avoid runway incursions/collisions. Simulation experiments showed that our system has long-range prediction of incidents, effective explicit instructions and/or suggestions, and flexible model for different aircraft and air traffic control policies. However, our current implementation cannot solve the problems of tardy, false or missed alerts caused by erroneous or missing sensory data.

Our future work will consider such factors such as missed/false surveillance information, extreme weather, as well as their consequences. The evaluation scenarios in this work are not adequate enough, thus more test scenarios are needed for a more complete test. Besides, we will consider use our system for other runway safety problems, such as runway excursion and runway confusion.

Chapter 7

Case study: Information security of computing services

7.1 Overview

Different information systems suffer different attacks. Although intrusion detection systems (IDS) make big progress on defending against computing malice, most current IDSs are general purpose, and usually weak in certain application systems/certain type of attacks. We argue that different information system should have different models of attack prediction/detection, and have ability to prevent/stop these attacks. Besides, ideal IDS should use explicit model and ability of customization. In this case study, we apply our methodology to a web server, which is a typical computing service. By using the methodology, we analyzed how to detect the events using for prediction and decision making, analyzed how to take the action by configuring the firewall, constructed anticipatory model, and built the target system. To evaluate the target system, we first took some real attacks in a real network, including DoS attacks (including HTTP DoS attacks), probe attacks (using Nmap), and web hacking attacks (including input validation, SQL query poisoning, directory browsing, and retrieving non-web files). The target system can predict/detect all these attacks and prevent these attacks. Besides, we also use KDD99, a popular dataset, to test the target system. The result by using KDD99 showed that the target system have low missed/false alarms.

7.2 Introduction

Today, information security of information systems is no longer about confidentiality, integrity and availability, but about ensuring that the systems are predictably dependable in the face of all sorts of malice [17]. It is difficult to defend a system against malice, because (1) there are various malicious behaviors for different systems/applications, (2) new vulnerabilities of systems are exposed, as well as new approaches for attacking are invented every day, (3) experienced attackers' behaviors are difficult to detect directly, (4) even if we could detect a malicious activity, it is difficult to choose appropriate actions to stop the ongoing malicious

activity and prevent future attacks by the system itself, and (5) the malice defense system itself is also the target of malice, and can be compromised.

To defend against computing malice, we argue an ideal malice defense system should (1) have the ability to detect malicious behaviors, (2) to predict imminent attacks, (3) to handle the malice by the system itself, (4) use explicit model (for the sake of explicable system behaviors), (5) handle application-level attacks, (6) have ability for customization/interoperability, (7) have the ability for update, (8) handle contextual/collective misuse/abnormal, (9) ensure its own security, and 10) function continuously and persistently.

As a mainstream approach to defend against malice, an intrusion detection system (IDS) is a device or software application that monitors network or system activities for malicious behaviors or policy violations, logs information about them, and reports them to security administrators, furthermore, some IDSs can attempt to stop detected possible attacks, also known as intrusion detection and prevention systems (IDPS) [156]. Although current IDSs make big progress on defending against computing malice, there is still a gap between them and ideal systems.

On the other hand, anticipatory reasoning-reacting systems (ARRS) [38] were proposed as high secure systems with the ability to defend against malice anticipatorily. Some features of ARRSs, such as logical reasoning based prediction/decision making and persistent computing [48], may contribute to an ideal malice defense system. However, until now, there is no concrete implementation of ARRS for security, as well as no evidence showing the practical usefulness of anticipatory computing for security.

As a step towards to ideal secure systems, we design and implement an ARRS for malice defense, which can adapt to different application systems by configuring different information source, anticipatory models, and anticipatory actions. We present the system design and implementation, and discuss how our system attempt to meet the features of ideal malice defense systems. We also evaluate out system by using KDD99 dataset and a case study of web server.

The contributions of this work are: (1) to propose what features ideal malice defense systems should have, (2) to analyze the gap between current IDSs and ideal malice defense systems, (3) to present why some advantages of ARRSs could contribute ideal secure systems, (4) to present and evaluate a practical implementation of ARRS for security, thus to show anticipatory computing's practical usefulness for security.

7.3 Ideal malice defense systems

To defend against computing malice, we argue ideal malice defense systems should have the following features.

- *Detecting*

  The system can detect an ongoing malicious activity carried out by both outsiders and insiders.

- *Predicting*

  The system can identify suspicious activity, such as reconnaissance, then predict an imminent attack, besides, the system should also predict fatal attacks from trivial attacks and other suspicious activities.

- *Active response to malice*

  The system can choose and take actions automatically to both stop an ongoing attacks and prevent an imminent attack anticipatorily.

- *Explicit model (explicable behaviors)*

  The system should use explicable model for detecting/predicting malice and choosing actions, thus the system's behaviors are explicable. On the other hand, opaque model contravenes some laws such as European data protection law [17].

- *Handling application-level attacks*

  The system can handle application level attacks. Application level attacks will become the most common computing attacks, moreover, different applications suffer different attacks.

- *Customization/Interoperability*

  The system can be customized easily for different applications and different scenarios. Besides, interoperability is one of foundations for customization.

- *Update*

  The system can update to handle new attack techniques.

- *Handling contextual/collective misuse/abnormal*

  The system should handle contextual/collective misuse/abnormal. It is not easy to detect experienced attackers' behaviors, unless the system can handle contextual/collective misuse/abnormal.

- *Security*

  The system can ensure its own security, both preventing against attacks from outsiders and malicious behaviors and incorrect operations from insiders.

- *Persistently continuous functioning*

  The system can function continuously anytime without stopping its reactions even when it is being maintained, upgraded, reconfigured, or it is being attacked.

Table 7.1: Comparing current IDSs with ideal malice defense systems

IDS	Pred-icting	Response	Explicit model	Application level attacks	Custo-mization	Interop-erability	Update	Contextual/ Collective	Security	Persistent
Haystack [161]	no	passive	yes	no	low	low	not spec	not spec	low	not spec
IDES [124]	no	passive	partial	not finished	low	low	not spec	not spec	low	not spec
W&S [170]	no	passive	yes	low	low	low	not spec	not spec	low	not spec
NSM [86]	no	passive	yes	no	low	low	not spec	not spec	low	not spec
NADIR [94]	no	passive	yes	not spec	low	low	not spec	not spec	low	not spec
Hyperview [53]	no	passive	no	not spec	low	lower	not spec	not spec	low	not spec
DIDS [162]	no	passive	yes	not finished	low	low	not spec	not spec	low	not spec
ASAX [76]	no	passive	yes	not spec	higher	higher	higher	not spec	low	not spec
USTAT [91]	no	passive	no	not spec	low	low	not spec	not spec	low	not spec
DPEM [111]	no	passive	no	low	low	low	low	not spec	low	not spec
IDIOT [114]	no	passive	no	low	high	higher	low	not spec	low	not spec
NIDES [16]	no	passive	partial	low	high	higher	high	not spec	low	not spec
GrIDS [165]	no	passive	no	no	low	low	low	no	low	not spec
CSM [177]	no	active	not spec	no	low	low	not spec	not spec	low	not spec
Janus [69]	no	active	no	low	low	low	not spec	not spec	low	not spec
JiNao [98]	no	passive	partial	not spec	low	low	not spec	not spec	low	not spec
EMERALD [147]	no	active	partial	low	low	high	not spec	not spec	moderate	not spec
Bro [139]	no	passive	partial	yes	low	low	high	not spec	higher	not spec
SNORT [11]	no	active	yes	no	low	high	high	no	not spec	not spec
Check Point [2]	no	active	yes	no	low	high	high	not spec	not spec	not spec
Cisco IDS [3]	no	passive	yes	no	low	low	higher	not spec	not spec	not spec
Cisco IPS [4]	no	active	yes	no	low	high	higher	not spec	not spec	not spec
Dragon [6]	no	passive	yes	no	low	high	high	no	low	not spec
Prelude [10]	no	passive	yes	low	low	high	not spec	no	low	not spec
Firestorm [7]	no	passive	yes	low	low	high	not spec	no	low	not spec
IBM SNIPS [8]	no	active	yes	Web	low	high	not spec	not spec	not spec	not spec
ARMD [122]	no	passive	yes	no	high	not spec	not spec	not spec	not spec	not spec
ISOA [179]	no	active	yes	high	high	high	low	not spec	not spec	not spec
NIDX [24]	no	passive	yes	no	not spec	low	not spec	not spec	not spec	not spec
NetSTAT [173]	no	passive	not spec	no	high	high	not spec	not spec	not spec	not spec
MADAM ID [117]	no	passive	yes	low	low	low	low	not spec	not spec	not spec
SPADE [164]	no	passive	no	no	no	no	no	no	lower	not spec
SmartSifter [181]	no	passive	no	no	no	no	no	no	lower	not spec
MET [27]	no	passive	no	Email	not spec	not spec	not spec	not spec	not spec	not spec
MINDS [60]	no	passive	no	not spec	not spec	not spec	not spec	not spec	not spec	not spec
Valdes [171]	no	passive	no	not spec	not spec	not spec	not spec	not spec	not spec	not spec
ADAM [22]	no	passive	no	not spec	not spec	not spec	low	not spec	not spec	not spec
IDDM [13]	no	passive	yes	not spec	low	not spec	low	not spec	not spec	not spec
PHAD [125]	no	passive	no	no	not spec	low	low	not spec	not spec	not spec
ADMIT [158]	no	passive	no	not spec	low	low	low	not spec	not spec	not spec
NetProbe [153]	no	active	yes	yes	high	high	high	not spec	high	not spec
eBayes [172]	no	passive	no	not spec	not spec	not spec	low	not spec	not spec	not spec
Zhao [184]	no	not spec	no	no	not spec	not spec	not spec	not spec	not spec	not spec
Kim [104]	no	passive	no	no	not spec	not spec	not spec	not spec	not spec	not spec
Munz [131]	no	passive	no	no	high	high	not spec	not spec	not spec	not spec
Lakhina [115]	no	passive	no	no	not spec	not spec	not spec	not spec	not spec	not spec
Wagner [175]	no	passive	no	no	not spec	not spec	not spec	not spec	not spec	not spec
Gates [67]	no	passive	no	no	not spec	not spec	not spec	not spec	not spec	not spec
Dubendorfer [55]	no	passive	no	no	high	high	not spec	not spec	not spec	not spec
Collins [50]	no	passive	no	no	not spec	not spec	not spec	not spec	not spec	not spec
Dressler [54]	no	passive	yes	no	no	not spec	low	not spec	not spec	not spec
Karasaridis [99]	no	passive	no	no	not spec	not spec	not spec	not spec	not spec	not spec
Livadas [166]	no	passive	no	no	not spec	not spec	not spec	not spec	not spec	not spec
BotMiner [74]	no	passive	no	no	not spec	not spec	not spec	not spec	not spec	not spec
D-SCIDS [12]	no	passive	no	not spec	not spec	not spec	not spec	not spec	not spec	not spec
Khan [102]	no	passive	no	not spec	not spec	not spec	not spec	not spec	not spec	not spec
Su [167]	no	passive	no	no	low	low	low	not spec	not spec	not spec
Horng [90]	no	passive	no	not spec	low	low	low	not spec	not spec	not spec
Cabrera [33, 34]	yes	active	yes	no	no	low	not spec	not spec	not spec	not spec

7.4 Current intrusion detection systems

In table 7.1, we list 59 intrusion detection systems (IDS), and try to compare them with ideal malice defense systems. Some conclusions in the table refer to [20, 116, 163].

By comparing compare the current IDSs with ideal malice defense systems, we have following conclusions.

- Most IDSs are passive.
- Almost all of active IDSs are directly reactive.
- Most IDSs do not consider to handle contextual/collective misuse/abnormal.
- Almost all active IDSs do not consider prediction.
- A lot of IDSs use opaque models.
- Most IDSs are week in customization/interoperability.
- Almost all network based IDS cannot handle encrypted application-level malice. Although using a proxy may solve this problem, it defeats application systems' efficiency.
- Most IDSs are general purpose. However, they are often weak for a certain application scenario.
- A lot of IDSs are difficult to change/update their models. They have to rebuild models for different application scenarios, and some of them also need training dataset which is difficult to obtain in fact.
- Most IDSs do not consider their own security, especially insiders' malice or mistakes.
- Almost all IDSs do not consider persistently continuous functioning.

7.5 Advantages of ARRSs for malice defense

7.5.1 Logical reasoning method

An ARRS uses logical reasoning to predict and make decisions of active response [38]. The basic idea of logical reasoning method is to explicitly separate the underlying logic system, reasoning/computing mechanism, and empirical knowledge in any prediction/decision making such that both underlying logic system and empirical knowledge can be revised/replaced/customized in various predictions/decision making processes performed by an area-independent, task-independent, general-purpose reasoning mechanism [44, 45].

The method of prediction/decision making by logic reasoning has the following characteristics [44, 45]. First, to explicitly separate the underlying logic system,

reasoning/computing mechanism, and empirical knowledge directly results in an adaptive prediction/decision making method such that one can easily develop, modify, and improve the underlying logic system, reasoning/computing mechanism, and empirical knowledge separately to satisfy different requirements from various application areas. Second, as the underlying logic system (i.e., logical validity criterion) and empirical knowledge are separated explicitly, the prediction/decision making results can be evaluated from two aspects: logical aspect that is area/problem independent and empirical aspect that is area/problem dependent. Third, as a characteristic of development and maintenance technology, the underlying logic system, reasoning/computing mechanism, and empirical knowledge can be developed and maintained separately. Fourth, because logic systems and their formal languages are adopted as the absolute, area/problem independent criterion of correctness/validity and representation languages, the approach of prediction/decision making by logic reasoning is more suitable to qualitative methods rather than quantitative methods.

7.5.2 Persistent computing

An ARRS is a persistent computing system, which is a reactive system that functions continuously anytime without stopping its reactions even when it is being maintained, upgraded, or reconfigured, it had some trouble, or it is being attacked [48]. Persistent computing systems have the two key characteristics and/or fundamental features: (1) persistently continuous functioning, i.e., the systems can function continuously and persistently without stopping its reactions, and (2) dynamically adaptive functioning, i.e., the systems can be dynamically maintained, upgraded, or reconfigured during its continuous functioning [48].

7.6 Applying the methodology

Phase 1: Analyze the target domain

The possible attacks to a computing service includes DoS, probing, compromises, and viruses/worms/Trojan horses [116]. The possible anticipatory actions include blocking access from the attacker or to the target, killing connections to or from a particular host, network, and port, changing an attack's content, terminating a network connection or a user session, and blocking access form certain hosts or user accounts. Besides, the system could notify security administrators notifications about important detections/predictions/responses about malicious activities.

Phase 2: Analyze the legacy system

The set I includes network traffic and network data. To get these traffic/data, there are two kinds of observers: inline observers and passive observers. An inline observer is deployed so that the network traffic/data transfer it is monitoring must pass through it, such as a hybrid firewall, a proxy, and a security gateway.

A passive observer is deployed so that it monitors a copy of the actual network traffic, NetFlow or sFlow, host system status, and application logs and status. The actions in set O are carried out by actuators. An actuator could be a firewall (either network firewall or host-based firewall), a router, or a switch, which can block access from the attacker or to the target; An actuator could be a program that kills connections to or from a particular host, network, and port, such as Tcpkill; An actuator could be a security proxy, which can change an attack's content; An actuator could also be an application service, while some application could terminate a network connection or a user session, block access form certain hosts or user accounts.

Phase 3: Define requirements of the target system

The target system can identify suspicious activity, such as reconnaissance, then predict an imminent attack, besides, the system should also predict fatal attacks from trivial attacks and other suspicious activities. The target system can choose and take actions automatically to both stop an ongoing attacks and prevent an imminent attack anticipatorily. The target system should use explicable model for detecting/predicting malice and choosing actions, thus the system's behaviors are explicable. The target system can handle application level attacks. The target system can be customized for different applications and different scenarios, while interoperability is one of foundations for customization.

Phase 4: Construct anticipatory model

We chose *temporal deontic relevant logics* [42] as the logic basis. We present preventing web server against HTTP DoS attacks as an example. In predictive model, "$\forall ip\_addr(MaxConnection(ip\_addr) \Rightarrow HttpDos(ip\_addr))$" means "if a host *ip_addr* makes more than max concurrent requests, it is taking a HTTP DoS attack.", and "$\forall ip\_addr(TooFrequent(ip\_addr) \Rightarrow HttpDos(ip\_addr))$" means "if a host *ip_addr* requests the same page more than a few times per second, it is taking a HTTP DoS attack." In behavioral model, $\forall ip\_addr(HttpDos(ip\_addr) \Rightarrow Block(ip\_addr))$ means "if host *ip_addr* is taking a HTTP DoS attack, block it."

Phase 5: Implementation

Because we also chose temporal deontic relevant logics as the logic basic. The interesting terms of predictions are predicates of attacks, such as $HttpDos$. The interesting terms of decision making are $Block$, $Kill$, $ChangeContent$, etc.

7.7 The implemented system

7.7.1 Overview

After applying the methodology, we implemented an ARRS with ability of malice defense, which can record information related to malice activities, detect/predict malice activities based on anticipatory model, then stop/prevent these malice activities, besides, the system can also notify security administrators of important detections/predictions/responses.

7.7.2 Architecture

Figure 7.1 shows the architecture of the ARRS for malice defense, which includes the following components.

- *LTDB* is a logical theorem database, which stores fragments of logical systems [38].
- *ETDB* is an empirical theory database storing anticipatory model, including *world model*, *predictive model*, and *behavioral model*, which express the real world, predictive laws and behavioral patterns of the target domain as empirical theories represented by logical formulas correspondingly.
- *RDB* is a rule database storing *filter rules*, *translation rules*, *interesting formula definitions*, *interesting terms*, and *actions mapping rules*.
- *Observers* monitor activities and status of the networks, host servers, or application services, then send these trivial sensory data to the filter. There are two kinds of observers: inline observers and passive observers. An inline observer is deployed so that the network traffic/data transfer it is monitoring must pass through it, such as a hybrid firewall, a proxy, and a security gateway. A passive observer is deployed so that it monitors a copy of the actual network traffic, NetFlow or sFlow, host system status, and application logs and status.
- *Filter* filters out the trivial sensory data and generates important and useful information for detection/prediction or decision making.
- *Formula generator* encodes the sensory data into logical formulas used by the forward reasoning engine according to the translation rules.
- *Forward reasoning engine* is a program to automatically draw new conclusions by repeatedly applying inference rules, to given premises and obtained conclusions until some previously specified conditions are satisfied [47]. The forward reasoning engine gets logical formulas translated at the formula generator, fragment of (a) logic system(s), and the anticipatory model, then it deduces predictions/next actions.
- *Formula chooser* chooses nontrivial conclusions, i.e., predictions and next actions, from forward reasoning engine according to interesting formula definitions and interesting terms.
- *Enactor* receives next actions from formula chooser, then gives corresponding instructions according to the actions mapping rules to actuators.
- *Actuators* receive instructions from the enactor, and carry out actions specified by the instructions to stop/prevent malicious behaviors. An actuator could be a firewall (either network firewall or host-based firewall), a router, or a switch, which can block access from the attacker or to the target; An actuator could be a program that kills connections to or from a particular host,

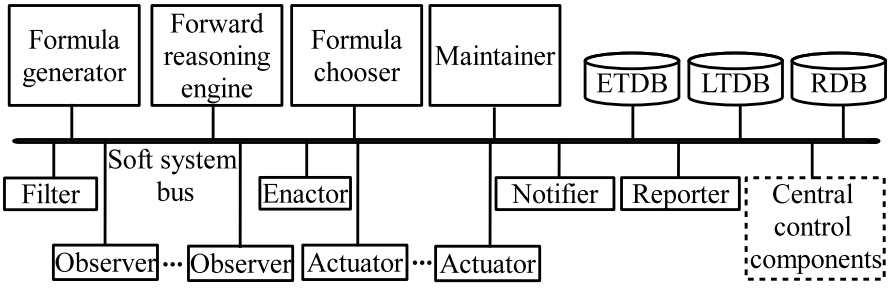

Figure 7.1: System architecture

network, and port, such as Tcpkill; An actuator could be a security proxy, which can change an attack's content; An actuator could also be an application service, while some application could terminate a network connection or a user session, block access form certain hosts or user accounts.

- *Notifier* notifies security administrators notifications about important detections/predictions/responses about malicious activities. The notifier gives notifications by e-mail, short message, as well as the system's user interface.

- *Reporter* summarizes the detections/predictions/responses about malicious activities, as well as provides details of this information.

- *Maintainer* checks whether there are contradictional theorems in ETDB against a newly added/modified empirical theorem, when we construct/update/modify the anticipatory model. If the contradictional theorems exist, the maintainer lists up theorems deduced from the contradictional theorems in ETDB.

- *Soft system bus* [39] and *central control components* provide a communication channel for each component and an infrastructure for persistent computing [39].

7.7.3 Mechanisms

In this subsection, we present the system's mechanisms of anticipation, persistent computing, reliability and security.

- *Anticipation*

  In order to detect/predict malicious activities then choose and take actions actively to stop/prevent the malice, our system adopt anticipatory approach based on logical reasoning method. Figure 7.2 shows the data flow diagram of anticipation. There are three steps for anticipation. First step is to detect and predict malicious activities. Second step is to choose anticipatory actions. And third step is to take these anticipatory actions to stop/prevent these malicious activities. For detecting/predicting malice, there are four steps. First, observers monitor activities and status of the networks, host

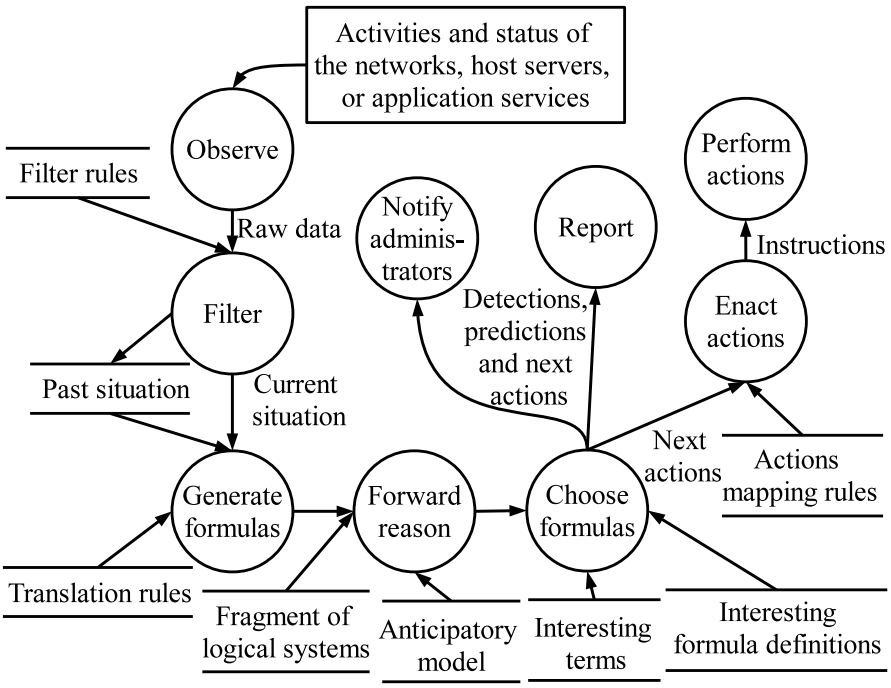

Figure 7.2: Data flow diagram of anticipation

servers, or application services, then send these trivial sensory data (current situation) to the filter, as well as store them. Second, formula generator translates the filtered current situation and past situation to logical formulas according to translation rules. Third, the forward reasoning engine gets input of these logical formulas about current/past situation, the world model, the predictive model, and fragment of logic system, then apply forward reasoning. Forth, the formula chooser chooses predictions/detections from all formulas reasoned out by the forward reasoning engine. The predictions/detections are used for decision making, as well as sent to notifier and reporter. For choosing next actions, it is similar with the process of prediction, where the main difference is to use behavior model instead of predictive model for logical reasoning. After the formula chooser chooses next actions, these next actions are sent to the enactor, notifier and reporter. For taking actions, the enactor receives these next actions and translate them to instructions, then gives the instructions to actuators. An actuator receives the instructions, then take the actions to stop/prevent the malice activities.

- *Persistent computing, reliability and security*

  The system's persistent computing, reliability and security mainly resort to soft system bus (SSB) methodology [39]. The SSB methodology provides a way that maintenance, upgrade, and reconfiguration of an SSB-based system [39] (i.e., adding, replacing, or removing functional components) can be done without stopping the running of the whole system. Besides, the SSB methodology provides a way to control and keep reliability and security in different degree at different points in an SSB-based system. In an

SSB-based system, we can anticipatorily protect the whole system by removing a functional component with some danger temporarily, or anticipatorily protect an important functional component by stopping its all interaction with other components. Moreover, the SSB methodology also provides information hiding, i.e., to make information not visible and not accessible to those program units or human objects that do not have rights to access the information. On the other hand, in order to realize features about handling different application-level attacks, customization, interoperability, and update, it is necessary to dynamically adding/upgrading observers and actuators. Persistent computing also provide a fundamental for dynamically adding/upgrading observers and actuators. In order to deal with insiders' malice/mistakes, such as adding some empirical knowledge which conflicts with the current knowledge in anticipatory model, the maintainer could check and detect that, then the system could notify this to other administrators or supervisors.

7.8 Evaluation

7.8.1 KDD99

The KDD99 dataset [1] was derived from the DARPA98 network traffic dataset [5] which is the most popular dataset that has ever been used in the intrusion detection field. We use KDD99 dataset to show logical reasoning method are at least not worse than other approaches for detecting malice activities. However, because almost all elements in KDD99 dataset are statistics but do not include contextual information and details (e.g., IP address, port number), we cannot predict based on KDD99 dataset, and we do not choose and take actions to defend these malice activities. Because elements in KDD99 dataset belong to filtered information, we directly input each element of KDD99 dataset to formula generator one by one. In this experiment, we only deal with some DoS and probe attacks, because those remote-to-local and user-to-root attacks are system-dependent. We use *temporal deontic relevant logics* [42] as the logic basis. Table 7.2 shows the experimental results.

7.8.2 A case study of web server

In order to show our system's practical use, we carry out a case study of web server. Figure 7.3 shows the experimental environment. The web server is Apache on Linux. The firewall is a Linux firewall. The client's OS are Linux used for generate attacks including DoS attacks (including HTTP DoS attacks), probe attacks (using Nmap), and web hacking attacks (including input validation, SQL query poisoning, directory browsing, and retrieving non-web files). The observers include one for monitoring network status, one for monitoring apache logs, and one for monitoring other system resource. The actuators include one for reconfiguring the firewall and one for terminating a connection between a client and the server. We use *temporal*

Table 7.2: Detection performance on the KDD99 test dataset

Attack name	Protocal type	Number of occurrences on test data	Number of occurrences detected	False classification	False alarm
Back	TCP	2203	2199	0	0
Ipsweep	TCP	94	91	0	0
Land	TCP	21	21	0	1
Neptune	TCP	107201	107095	8	0
Nmap	TCP	103	103	0	0
Nmap	UDP	25	24	0	0
Teardrop	UDP	979	979	0	0
Ipsweep	ICMP	1153	1153	104	0
Nmap	ICMP	103	103	6	0
Pod	ICMP	264	264	0	2
Smurf	ICMP	280790	280790	0	0

deontic relevant logics [42] as the logic basis. The system can detect/predict the attacks mentioned above, and prevent these attacks automatically by blocking a certain client (by adding rules to firewall or terminating a connection by sending resets, and terminating corresponding Apache processes).

We show an example about how our system handles HTTP DoS attacks. In order to handle HTTP DoS attacks, we have following empirical theories in the anticipatory model. In predictive model (the rules refer to [9]), "$MaxConnection(ip\_addr) \Rightarrow HttpDos(ip\_addr)$" means "if a host *ip_addr* makes more than max concurrent requests, it is taking a HTTP DoS attack.", and "$TooFrequent(ip\_addr) \Rightarrow HttpDos(ip\_addr)$" means "if a host *ip_addr* requests the same page more than a few times per second, it is taking a HTTP DoS attack." In behavioral model, $HttpDos(ip\_addr) \Rightarrow Block(ip\_addr)$ means "if host *ip_addr* is taking a HTTP DoS attack, block it." To observe requisite events, we utilize "netstat" to monitor concurrent requests, and analyze apache log to monitor visit frequency. To realize blocking action, the enactor gives instruction "iptables -I INPUT -s *ip_addr* -j DROP" to the firewall, and kills all Apache processes connecting with *ip_addr* by SIGKILL.

7.9 Summary

We proposed what features ideal malice defense systems should have, analyzed the gap between current intrusion detection systems (IDS) and ideal malice defense systems, presented why some advantages of anticipatory reasoning-reacting systems (ARRS) could contribute ideal secure systems, presented and evaluated a practical implementation of ARRS for malice defense. We showed how our system attempts to satisfied the features of ideal malice defense systems, except for handling contextual/collective misuse/abnormal.

Our system is still in its infancy. For KDD99 dataset, we cannot predict malice

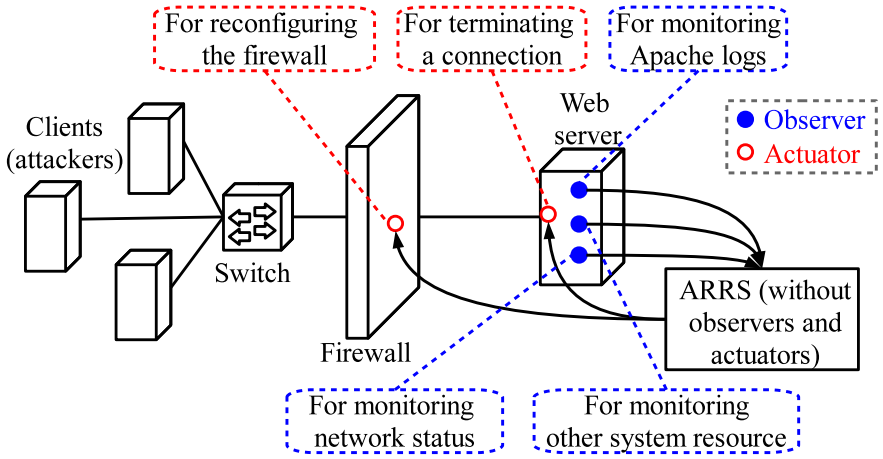

Figure 7.3: The experimental environment of ARRS for web server

activities and take actions to defend these malice activities because of the limitation of KDD99 dataset. For the case study of web server, we only handle some traditional attacks. Until now, we do not have evidence to show that our system performs better than other approaches from the aspect of detecting malice activities, makes significant and practical prediction about malice activities, and handles practical contextual/collective misuse/abnormal.

There are several future works: to prove anticipatory computing's practical usefulness on handling contextual/collective misuse/abnormal, to make significant and practical prediction for some typical malice activities, and to apply our system on other application areas. Furthermore, in order to make our system practical useful, we will build some optional anticipatory models, observers and actuators for some common applications (such as web servers and database systems), thus the users could easily build their own system by choosing and using these prepared models and components.

Chapter 8

Discussion

Although there are some particularities in the three case studies, we can still argue our methodology is a general one.

First, our case studies showed that our methodology can deal different kinds of reactive systems. A reactive system can have following three kinds of functions [178].

- Informative: To provide information about the subject domain, such as to answer questions and produce reports.
- Directive: To direct the entities in the subject domain, such as to control or guide a physical entity.
- Manipulative: To manipulate lexical items in the subject domain, such as to create, remove, or modify lexical items.

The legacy reactive systems in first and second case studies have directive and informative functions, while the legacy reactive system in third case study has informative and manipulative functions. Therefore, from viewpoint of system function, our case studies covers different kinds of reactive systems, thus, our methodology is general.

Second, our case studies showed that our methodology can improve both the legacy system's safety and security. *Safety* is freedom of risk [52] (accidents or losses) [120]. *Security* is prevention of or protection against access to information by unauthorized recipients or intention but unauthorized destruction or alteration of that information [52]. Safety and security are closely related and similar, while the important differences between them is that security focuses on malicious actions. Our first and second case studies improve the system's safety by avoiding disasters/accidents. Our third case study improve the system's security by preventing attacks/ malicious behaviors.

The main particularity among the three case studies is quantitative calculating of the action planer. The quantitative calculating for each case study is special, while it cannot be used in other cases. At present, we can eliminate this particularity by implementing different action planners for different target domains.

Chapter 9

Conclusions

9.1 Conclusions

It is difficult to ensure reactive systems' safety and security, because the problem is not to ensure what should happen, but to ensure what should not happen [17, 120]. Furthermore, it is more challenging to improve existing reactive systems' safety and security.

We have proposed a new approach by extending the systems with anticipatory ability without reimplementing the whole of the existing system. The proposed methodology can be used to extend various reactive systems with anticipatory ability, but preserve the system's original functions. We presented three case studies to show how to apply our methodology to legacy reactive systems. We also discussed the generality and particularity of the case studies.

We can conclude: (1) it is possible to extend an existing reactive system with anticipatory ability, (2) the anticipatory ability is useful to the legacy reactive system, thus such an extension is rewarding, and (3) our method is effective, at least in the three case studies.

9.2 Contributions

This work has following contributions. First, we conceived a new approach to improve existing reactive systems safety and/or security by extending the system with anticipatory ability. Second, we proposed a general methodology, which can extend various reactive systems with anticipatory ability, and we showed the effectiveness of the methodology by applying the methodology to different case studies. Third, we built three practical ARRSs in the case studies, thus showed the practical usefulness of anticipation and ARRSs for safety and security. Previous studies of ARRSs are mainly theoretical, such as formal definition, architecture design, mechanism of prediction and decision-making, and prototype implementation, thus, there was a gap between those theoretical work and practical ARRSs. Whereas, in this work, we built practical ARRSs, as well as solve several practical problems when building practical ARRSs.

9.3 Future works

To prove the generality of our proposed methodology, the best way is to apply our methodology to various target domains. Thus, our future work is to find out some other interesting areas, which require high safety or high security, then extends some existing critical reactive systems in these areas with anticipatory ability.

When we construct/update/modify the anticipatory model, it is necessary to check whether there are contradictional theorems in the current anticipatory model against a newly added/modified empirical theorem. A truth maintenance system can check whether there are contradictional theorems in anticipatory model against a newly added/modified empirical theorem, when we update/modify the anticipatory model.

A software development framework as a reusable, semi-finished application that can be customized by developers to produce custom applications, is an ideal solution to build complex software for different target domains, such as ARRSs. By now, we have proposed the design of such a framework in our paper "a software development framework for various anticipatory reasoning-reacting systems".

Publications

Refereed papers published in journals or books (first author)

- Kai Shi, Yuichi Goto, Zhiliang Zhu, and Jingde Cheng, "Making Existing Reactive Systems Anticipatory," in R. Lee, et al. (Ed.), "Computer and Information Science 2013," Studies in Computational Intelligence, Vol. 493, pp.17-32, Springer-Verlag, June 2013.
- Kai Shi, Yuichi Goto, Zhiliang Zhu, and Jingde Cheng, "Anticipatory Emergency Elevator Evacuation Systems," in A. Selamat, N. T. Nguyen, and H. Haron (Eds.), "Intelligent Information and Database Systems, 5th Asian Conference, ACIIDS 2013, Kuala Lumpur, Malaysia, March 18-20, 2013, Proceedings, Part I," Lecture Notes in Artificial Intelligence, Vol. 7802, pp. 117-126, Springer-Verlag, March 2013.
- Kai Shi, Kazunori Wagatsuma, Yuichi Goto, and Jingde Cheng, "World Model, Predictive Model, and Behavioral Model of an Anticipatory Reasoning-Reacting System for Runway Incursion Prevention, " International Journal of Computing Anticipatory Systems, CHAOS, 2012 (in press).
- Kai Shi, Yuichi Goto, and Jingde Cheng, "A Software Development Framework for Various Anticipatory Reasoning-Reacting Systems," International Journal of Computing Anticipatory Systems, CHAOS, 2012 (in press).
- Kai Shi, Yuichi Goto, Zhiliang Zhu, and Jingde Cheng, "Anticipatory Runway Incursion Prevention Systems, IEICE Transactions on Information and Systems, Vol. E96-D, No. 11, IEICE, November 2013 (to appear).

Refereed papers published in international conference proceedings (first author)

- Kai Shi, Bo Wang, Yuichi Goto, Zhiliang Zhu, and Jingde Cheng, "An Anticipatory Reasoning-Reacting System for Defending Against Malice Anticipatorily," Proc. 4th IEEE International Conference on Software Engineering and Service Science (ICSESS 2013), pp. 732-737, Beijing, China, IEEE press, May 2013 (Acceptance rate: 28.8%).

Refereed papers published in journals or books (co-author)

- Hongbiao Gao, Kai Shi, Yuichi Goto, and Jingde Cheng, "Finding Theorems in NBG Set Theory by Automated Forward Deduction Based on Strong Relevant Logic," in D.-Z. Du and G. Zhang (Eds.), "Computing and Combinatorics, The 19th Annual International Conference, COCOON 2013, Hangzhou, China, June 21-23, 2013, Proceedings," Lecture Notes in Computer Science, Vol. 7936, pp. 697-704, Springer-Verlag, June 2013.

Refereed papers published in international conference proceedings (co-author)

- Hongbiao Gao, Kai Shi, Yuichi Goto, and Jingde Cheng, "Automated Theorem Finding by Forward Deduction Based on Strong Relevant Logic: A Case Study in NBG Set Theory," Proc. 11th International Conference on Machine Learning and Cybernetics, pp. 1859-1865, Xi'an, China, The IEEE Systems, Man, and Cybernetics Society, July 2012.
- Liqing Xu, Kai Shi, Yuichi Goto, and Jingde Cheng, "ISEC: An Information Security Engineering Cloud," Proc. 3rd IEEE International Conference on Software Engineering and Service Science, pp. 750-753, Beijing, China, IEEE Press, June 2012.
- Gefei Sun, Kenichi Yajima, Junichi Miura, Kai Shi, Yuichi Goto, and Jingde Cheng, "A Supporting Tool for Creating and Maintaining Security Targets According to ISO/IEC 15408," Proc. 3rd IEEE International Conference on Software Engineering and Service Science, pp. 745-749, Beijing, China, IEEE Press, June 2012.
- Bo Wang, Kai Shi, Yuichi Goto, and Jingde Cheng, "Generation of System Dependence Nets for Ada 2005 Programs," Proc. 2nd IEEE International Conference on Computer Science and Automation Engineering, Vol. 3, pp. 401-406, Zhangjiajie, China, IEEE Press, May 2012.

Bibliography

[1] KDD cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.

[2] Check Point. http://www.checkpoint.com/products/ips-1/, 2012.

[3] Cisco IDS. http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/index.shtml, 2012.

[4] Cisco IPS. http://www.cisco.com/en/US/products/ps5729/Products_Sub_Category_Home.html, 2012.

[5] DARPA intrusion detection data sets. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html, 2012.

[6] Dragon. http://www.intrusion-detection-system-group.co.uk/dragon.htm, 2012.

[7] Firestorm. http://www.scaramanga.co.uk/firestorm/, 2012.

[8] IBM Security Network Intrusion Prevention System. http://www-01.ibm.com/software/tivoli/products/security-network-intrusion-prevention/, 2012.

[9] mod_evasive. http://www.zdziarski.com/blog/?page_id=442, 2012.

[10] Prelude. http://www.prelude-technologies.com, 2012.

[11] Snort. http://www.snort.org, 2012.

[12] A. Abraham, R. Jain, J. Thomas, and S.Y. Han. D-SCIDS: distributed soft computing intrusion detection system. *Journal of Network and Computer Applications*, 30(1):81–98, 2007.

[13] T. Abraham. IDDM: Intrusion detection using data mining techniques. Technical Report DSTO-GD-0286, DSTO Electronics and Surveillance Research Laboratory, Department of Defense, Australia, 2001.

[14] L. Aceto, A. Ingólfsdóttir, K.G. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.

[15] Aircraft Accident Investigation Bureau. Final report No. 2113 by the Aircraft Accident Investigation Bureau, 2010.

[16] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion detection expert system (NIDES): A summary. Technical Report SRI-CSL-95-07, CSL, SRI International, 1995.

[17] R. J. Anderson. *Security engineering: a guide to building dependable distributed systems*. Wiley, second edition, 2008.

[18] C André. Representation and analysis of reactive behaviors: A synchronous approach. In *IEEE-SMC'96, Computational Engineering in Systems Applications*, 1996.

[19] A. Attoui and M. Schneider. An object oriented model for parallel and reactive systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 84–93. IEEE, 1991.

[20] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers University, 2000.

[21] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. *Theory of Cryptography*, pages 336–354, 2004.

[22] D. Barbara, N. Wu, and S. Jajodia. Detecting novel network intrusions using bayes estimators. In *Proc. 1st SIAM Conference on Data Mining*. SIAM, 2001.

[23] G. C. Barney. Up-peak, down-peak & interfloor performance. *Elevator World*, 47:100–103, 1999.

[24] D.S. Bauer and M.E. Koblentz. NIDX-an expert system for real-time network intrusion detection. In *Proc. 1988 Computer Networking Symposium*, pages 98–106. IEEE, 1988.

[25] A. Benveniste and G. Berry. Another look at real-time programming. In *Special Section of the Proceedings of the IEEE*, volume 79, 1991.

[26] G. Berry. The foundations of esterel. *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 425–454, 2000.

[27] M. Bhattacharyya, S. Hershkop, and E. Eskin. MET: An experimental system for malicious email tracking. In *Proc. 2002 workshop on New security paradigms*, pages 3–10. ACM, 2002.

[28] C. Blaess, C. Tsiampalidis, and J. Vallee. An application of artificial intelligence for the safety in the neighbourhood of airport runways. In *Proc. IEEE ICTAI'96 Workshop on Artificial Intelligence for Aeronautics and Space, Toulouse, France*. IEEE, Nov. 1996.

[29] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In *Proceedings of the 5th international conference on Automated technology for verification and analysis*, pages 162–176. Springer-Verlag, 2007.

[30] R. W. Bukowski. Emergency egress strategies for buildings. In *Proc. 11th International interflam conference*, pages 159–168, 2007.

[31] R. W. Bukowski. International applications of elevators for fire service access and occupant egress in fires. *CTBUH Journal*, 2010 Issue III:28–584, 2010.

[32] R. W. Bukowski. Addressing the needs of people using elevators for emergency evacuation. *Fire Technology*, 48(1):127–136, 2012.

[33] J.B.D. Cabrera, L. Lewis, X. Qin, W. Lee, and R.K. Mehra. Proactive intrusion detection and distributed denial of service attacks-a case study in security management. *Journal of Network and Systems Management*, 10(2):225–254, 2002.

[34] J.B.D. Cabrera, L. Lewis, X. Qin, W. Lee, R.K. Prasanth, B. Ravichandran, and R.K. Mehra. Proactive detection of distributed denial of service attacks using MIB traffic variables-a feasibility study. In *Proc. 2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings*, pages 609–622. IEEE, 2001.

[35] C. Campbell, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Model-based testing of object-oriented reactive systems with spec explorer. *Microsoft Research, MSR-TR-2005-59*, 2005.

[36] R. Cassell, C. Evers, J. Esche, and B. Sleep. NASA runway incursion prevention system (RIPS) Dallas-Fort Worth demonstration performance analysis. Technical Report CR-2002-211677, NASA, 2002.

[37] P. Checkland. *System thinking, system practice*. John Wiley & Sons, 1981.

[38] J. Cheng. Temporal relevant logic as the logical basis of anticipatory reasoning-reacting systems. In *Proc. Computing Anticipatory Systems: CASYS - 6th International Conference, AIP Conference Proceedings*, volume 718, pages 362–375. AIP, 2004.

[39] J. Cheng. Connecting components with soft system buses: A new methodology for design, development, and maintenance of reconfigurable, ubiquitous, and persistent reactive systems. In *Proc. 19th International Conference on Advanced Information Networking and Applications*, pages 667–672. IEEE Computer Society Press, 2005.

[40] J. Cheng. Relevant logic as the logical basis for decision making based on anticipatory reasoning. In *Proc. 2006 IEEE Annual International Conference on Systems, Man, and Cybernetics*, pages 1036–1041. The IEEE Systems, Man, and Cybernetics Society, 2006.

[41] J. Cheng. Strong relevant logic as the universal basis of various applied logics for knowledge representation and reasoning. 136:310–320, 2006.

[42] J. Cheng. Temporal deontic relevant logic as the logical basis for decision making based on anticipatory reasoning. In *Proc. 2006 IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1036–1041. IEEE, 2006.

[43] J. Cheng. Persistent computing systems based on soft system buses as an infrastructure of ubiquitous computing and intelligence. *Journal of Ubiquitous Computing and Intelligence*, 1(1):35–41, 2007.

[44] J. Cheng. Adaptive prediction by anticipatory reasoning based on temporal relevant logic. In *Proc. 8th International Conference on Hybrid Intelligent Systems*, pages 410–416. IEEE Computer Society Press, 2008.

[45] J. Cheng. Adaptive decision making by reasoning based on relevant logics. In *Proc. Computational Intelligent: Foundations and Applications, 9th International FLINS Conference*, pages 541–546. World Scientific, 2010.

[46] J. Cheng, Y. Goto, and N. Kitajima. Anticipatory reasoning about mobile objects in anticipatory reasoning-reacting systems. In *Proc. Computing Anticipatory Systems: CASYS - 8th International Conference, AIP Conference Proceedings*, volume 1051, pages 244–254. AIP, 2008.

[47] J. Cheng, S. Nara, and Y. Goto. FreeEnCal: A forward reasoning engine with general-purpose. In *Proc. 11th International on Conference Knowledge-Based Intelligent Information and Engineering Systems, Part II, Lecture Notes in Artificial Intelligence*, volume 4693, pages 444–452. Springer-Verlag, 2007.

[48] J. Cheng and F. Shang. Persistent computing systems as an infrastructure of computing anticipatory systems. *International Journal of Computing Anticipatory Systems*, 18:61–74, 2006.

[49] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Logics of Programs*, pages 52–71, 1982.

[50] M. Collins and M. Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *Proc. 10th International Workshop on the Recent Advances in Intrusion Detection, Lecture Notes in Computer Science*, volume 4637, pages 276–295. Springer, 2007.

[51] CTBUH. *Emergency evacuation elevator systems guideline.* CTBUH, 2004.

[52] J. Daintith and E. Wright. *A dictionary of computing.* Oxford University Press, sixth edition, 2008.

[53] H. Debar, M. Becker, and D. Siboni. A neural network component for an intrusion detection system. In *Proc. 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 240–250. IEEE, 1992.

[54] F. Dressler, W. Jaegers, and R. German. Flow-based worm detection using correlated honeypot logs. In *Proc. 2007 ITG/GI Symposium Communication in Distributed Systems (KiVS)*, pages 1–6. VDE-Verlag, 2007.

[55] T. Dubendorfer, A. Wagner, and B. Plattner. A framework for real-time worm attack detection and backbone monitoring. In *Proc. 1st IEEE International Workshop on Critical Infrastructure Protection*, page 10pp. IEEE, 2005.

[56] MD Edwards and D. Aspinall. The synthesis of digital systems using asm design techniques. *Computer Hardware Description Languages and their Applications*, pages 55–64, 1983.

[57] E. Emerson. Automated temporal reasoning about reactive systems. *Logics for Concurrency*, pages 41–101, 1996.

[58] E.A. Emerson and E.M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer programming*, 2(3):241–266, 1982.

[59] C.A. Ericson II. Fault tree analysis - a history. In *Proc. 17th International System Safety Conference*, 1999.

[60] L. Ertoz, E. Eilertson, A. Lazarevic, P.N. Tan, V. Kumar, J. Srivastava, and P. Dokas. Minds-minnesota intrusion detection system. In *Next Generation Data Mining*, pages 199–218. AAAI, 2004.

[61] Federal Aviation Administration. Fact sheet - airport movement area safety system. http://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=6302, accessed Feb. 11. 2013.

[62] Federal Aviation Administration. May 29, 2009 OE/D operational error category A. http://www.faa.gov/airports/runway_safety/videos/media/simulation.html, accessed Feb. 26. 2013.

[63] Federal Aviation Administration. *Pilot's handbook of aeronautical knowledge - appendix 1, runway Incursion Avoidance*. FAA, 2012.

[64] Federal Aviation Administration Airport Engineering Division. Runway status lights system. Technical Report Engineering Brief 64D, FAA, 2011.

[65] AB Ferrentino and HD Mills. State machines and their semantics in software engineering. In *Proc. IEEE COMPSAC'77 Conference*, pages 242–251, 1977.

[66] S. Feyock. Transition diagram-based cai/help systems. *International Journal of Man-Machine Studies*, 9(4):399–413, 1977.

[67] C. Gates, J.J. McNutt, J.B. Kadane, and M.I. Kellner. Scan detection on very large networks using logistic regression modeling. In *Proc. 11th IEEE Symposium on Computers and Communications*, pages 402–408. IEEE, 2006.

[68] J.L. Gertz and R.D. Grappel. Surveillance improvement algorithms for airport surface detection equipment model X (ASDE-X) at Dallas-Fort Worth airport. Technical Report ATC-333, MIT Lincoln Laboratory, 2007.

[69] I. Goldberg, D. Wagner, R. Thomas, and E.A. Brewer. A secure environment for untrusted helper applications (confining the wily hacker). In *Proc. 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography-Volume 6*, pages 1–1. USENIX, 1996.

[70] Y. Goto and J. Cheng. A transformation mechanism between sensory data and logical formulas for anticipatory reasoning-reacting systems. *International Journal of Computing Anticipatory Systems*, 2013.

[71] Y. Goto, R. Kuboniwa, and J. Cheng. Development and maintenance environment for anticipatory reasoning-reacting systems. *International Journal of Computing Anticipatory Systems*, 24:61–72, 2011.

[72] Y. Goto, S. Nara, and J. Cheng. Efficient anticipatory reasoning for anticipatory systems with requirements of high reliability and high security. *International Journal of Computing Anticipatory Systems*, 14:156–171, 2004.

[73] N.E. Groner. Selecting strategies for elevator evacuations. In *Proc. 2nd Symposium on Elevators, Fire, and Accessibility*, pages 186–189. ASME, 1995.

[74] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proc. 17th USENIX Security Symposium*, pages 139–154. USENIX, 2008.

[75] B. Guttman and E.A. Roback. *An introduction to computer security: the NIST handbook*. DIANE Publishing, 1995.

[76] N. Habra, B.L. Charlier, A. Mounji, and I. Mathieu. ASAX: Software architecture and rule-based language for universal audit trail analysis. In *Proc. 2nd European Symposium on Research in Computer Security, Lecture Notes in Computer Science*, volume 648, pages 435–450. Springer-Verlag, 1992.

[77] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Publishers, 1993.

[78] N. Halbwachs. Synchronous programming of reactive systems - a tutorial and commented bibliography. In *Computer Aided Verification*, pages 1–16. Springer, 1998.

[79] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[80] M. Hamilton and S. Zeldin. Higher order software - a methodology for defining software. *Software Engineering, IEEE Transactions on*, (1):9–32, 1976.

[81] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[82] D. Harel. Statecharts in the making: a personal account. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 5–1–5–43. ACM, 2007.

[83] D. Harel and E. Gery. Executable object modeling with statecharts. *Computer*, 30(7):31–42, 1997.

[84] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *Software Engineering, IEEE Transactions on*, 16(4):403–414, 1990.

[85] D. Harel and A. Pnueli. On the development of reactive systems. *Logics and models of concurrent systems*, pages 477–498, 1985.

[86] L.T. Heberlein, G.V. Dias, K.N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proc. 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 296–304. IEEE, 1990.

[87] R. Helm, I.M. Holland, and D. Gangopadhyay. Contracts: specifying behavioral compositions in object-oriented systems. In *OOPSLA/ECOOP '90 Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*, pages 169–180. ACM, 1990.

[88] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[89] L. Hoffmann. Amir pnueli: ahead of his time. *Communications of the ACM*, 53(1):22–23, 2010.

[90] S.J. Horng, M.Y. Su, Y.H. Chen, T.W. Kao, R.J. Chen, J.L. Lai, and C.D. Perkasa. A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert Systems with Applications*, 38(1):306–313, 2011.

[91] K. Ilgun. Ustat: A real-time intrusion detection system for unix. In *Proc. 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 16–28. IEEE, 1993.

[92] International Civil Aviation Organization. *Advanced surface movement guidance and control systems (A-SMGCS) manual*. ICAO, first edition, 2004. Doc 9830 AN/452.

[93] ISO/IEC. *ISO/IEC 19501:2005: Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2*. ISO, 2005.

[94] K. Jackson, D. DuBois, and C. Stallings. An expert system application for network intrusion detection. In *Proc. 14th National Computer Security Conference*, pages 215–225. NIST-NCSC, 1991.

[95] R.J.K. Jacob. Using formal specifications in the design of a human-computer interface. *Communications of the ACM*, 26(4):259–264, 1983.

[96] H. Jarvinen, R. Kurki-Suonio, M. Sakkinen, and K. Systa. Object-oriented specification of reactive systems. In *Software Engineering, 1990. Proceedings., 12th International Conference on*, pages 63–71. IEEE, 1990.

[97] O.H. Jensen and R. Milner. Bigraphs and mobile processes (revised). *Technical report, University of Cambridge Computer Laboratory*, 2004.

[98] Y.F. Jou, F. Gong, C. Sargor, S.F. Wu, and R. Cleaveland. Architecture design of a scalable intrusion detection system for the emerging network infrastructure. Technical Report CDRL A005, Dept. of Computer Science, North Carolina State University, 1997.

[99] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proc. 1st Workshop on Hot Topics in Understanding Botnets*, pages 1–8. USENIX, 2007.

[100] K. Kawai. Fire control system for elevator, 2008. US Patent 7,413,059.

[101] K. Kawai. Evacuation system and method for elevator control using number of people remaining, 2009. US Patent 7,637,354.

[102] L. Khan, M. Awad, and B. Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB Journal*, 16(4):507–521, 2007.

[103] J. H. Kim and B. R. Moon. Adaptive elevator group control with cameras. *IEEE Transactions on Industrial Electronics*, 48(2):377–382, 2001.

[104] M.S. Kim, H.J. Kong, S.C. Hong, S.H. Chung, and J.W. Hong. A flow-based method for abnormal network traffic detection. In *Proc. 2004 IEEE/IFIP Network Operations and Management Symposium*, volume 1, pages 599–612. IEEE, 2004.

[105] N. Kitajima, Y. Goto, and J. Cheng. Development of a decision-maker in an anticipatory reasoning-reacting system for terminal radar control. In *Proc. 4th International Conference on Hybrid Artificial Intelligence Systems, Lecture Notes in Artificial Intelligence*, volume 5572, pages 68–76. Springer-Verlag, 2009.

[106] N. Kitajima, S. Nara, Y. Goto, and J. Cheng. A deontic relevant logic approach to reasoning about actions in computing anticipatory systems. *International Journal of Computing Anticipatory Systems*, 20:177–190, 2008.

[107] N. Kitajima, S. Nara, Y. Goto, and J. Cheng. Fast qualitative reasoning about actions for computing anticipatory systems. In *Proc. 3rd International Conference on Availability, Reliability and Security*, pages 171–178. IEEE Computer Society Press, 2008.

[108] J. H. Klote. A method for calculation of elevator evacuation time. *Journal of fire protection engineering*, 5(3):83–95, 1993.

[109] J. H. Klote, B. M. Levin, and N. E. Groner. Feasibility of fire evacuation by elevators at FAA control towers. NISTIR 5445, National Institute of Standards and Technology, 1994.

[110] J. H. Klote, B. M. Levin, and N. E. Groner. Emergency elevator evacuation systems. In *Proc. 2nd Symposium on Elevators, Fire, and Accessibility*, pages 131–149. ASME, 1995.

[111] C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proc. 10th Annual Computer Security Applications Conferenc*, pages 134–144. IEEE, 1994.

[112] G. Kotonya and I. Sommerville. *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc.

[113] E. D. Kuligowski and R. W. Bukowski. Design of occupant egress systems for tall buildings. In *CIB World Building Congress*, volume 2004, 2004.

[114] S. Kumar and E.H. Spafford. A pattern matching model for misuse intrusion detection. In *Proc. 17th National Computer Security Conference*, pages 11–21. NIST-NCSC, 1994.

[115] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. *ACM SIGCOMM Computer Communication Review*, 35(4):217–228, 2005.

[116] A. Lazarevic, V. Kumar, and J. Srivastava. Intrusion detection: A survey. *Managing Cyber Threats*, pages 19–78, 2005.

[117] W. Lee and S.J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4):227–261, 2000.

[118] P. LeGuernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with signal. *Proceedings of the IEEE*, 79(9):1321–1336, 1991.

[119] J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. *CONCUR 2000Concurrency Theory*, pages 243–258, 2000.

[120] N. G. Leveson. *Safeware: system safety and computers*. Addison-Wesley, 1995.

[121] B. M. Levin and N. E. Groner. Some control and communication considerations in designing an emergency elevator evacuation system. In *Proc. 2nd Symposium on Elevators, Fire, and Accessibility*, pages 190–193. ASME, 1995.

[122] J.L. Lin, X.S. Wang, and S. Jajodia. Abstraction-based misuse detection: High-level specifications and adaptable strategies. In *Proc. 11th IEEE Computer Security Foundations Workshop*, pages 190–201. IEEE, 1998.

[123] P. B. Luh, B. Xiong, and S. C. Chang. Group elevator scheduling with advance information for normal and emergency modes. *IEEE Transactions on Automation Science and Engineering*, 5(2):245–258, 2008.

[124] T.F. Lunt, R. Jagannathan, R. Lee, S. Listgarten, D.L. Edwards, P.G. Neumann, H.S. Javitz, and A. Valdes. IDES: The enhanced prototype, a real-time intrusion-detection expert system. Technical Report SRI-CSL-88-12, CSL, SRI International, 1988.

[125] M.V. Mahoney and P.K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proc. 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 376–385. ACM, 2002.

[126] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification*, volume 1. Springer, 1992.

[127] F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. In *CONCUR'92*, pages 550–564. Springer, 1992.

[128] R. Milner. Bigraphical reactive systems: basic theory. Technical report, Technical Report 523, Computer Laboratory, University of Cambridge, 2001.

[129] R. Milner, R. Milner, R. Milner, and R. Milner. *A calculus of communicating systems*, volume 92. Springer-Verlag Germany, 1980.

[130] B. Morris and L. A. Jackman. An examination of fire spread in multi-storey buildings via glazed curtain wall facades. *Structural Engineer*, 81(9):22–26, 2003.

[131] G. Munz and G. Carle. Real-time analysis of flow data for network attack detection. In *Proc. 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 100–108. IEEE, 2007.

[132] S. Nara, F. Shang, T. Omi, Y. Goto, and J. Cheng. An anticipatory reasoning engine for anticipatory reasoning-reacting systems. *International Journal of Computing Anticipatory Systems*, 18:225–234, 2006.

[133] National Air Traffic Controllers Association National Office. Boston Logan controller's Nov. 24 save. http://www.youtube.com/watch?v=V-dbEYk-ikU, accessed Feb. 26. 2013.

[134] S. Nihei, H. Miyazaki, T. Koga, H. Aoyama, Y. Kakubari, and I. Yamada. Development of advanced-surface movement guidance and control (A-SMGC) systems (in Japanese). *Technical report of IEICE. SANE*, 109(397):101–106, 2010.

[135] International Civil Aviation Organization. *Manual on the Prevention of Runway Incursions*. ICAO, first edition, 2007. Doc 9870 AN/463.

[136] D.L. Parnas. On the use of transition diagrams in the design of a user interface for an interactive computer system. In *Proceedings of the 1969 24th national conference*, pages 379–385. ACM, 1969.

[137] L. Parrini, P. A. Spiess, K. Schuster, L. Finschi, P. Friedli, et al. Method and system for emergency evacuation of building occupants and a method for modernization of an existing building with said system, 2007. US Patent 7,182,174.

[138] C. Passerone, C. Sansoe, L. Lavagno, R. McGeer, J. Martin, R. Passerone, and A. Sangiovanni-Vincentelli. Modeling reactive systems in java. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 3(4):515–523, 1998.

[139] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer networks*, 31(23):2435–2463, 1999.

[140] J.L. Peterson. Petri net theory and the modeling of systems. *PRENTICE-HALL, INC., ENGLEWOOD CLIFFS, NJ 07632, 1981, 290*, 1981.

[141] B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. *Electronic Notes in Theoretical Computer Science*, 32:59–77, 2000.

[142] B. Pfitzmann, M. Schunter, and M. Waidner. *Secure reactive systems*. IBM Research Division, 2000.

[143] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 184–200. IEEE, 2001.

[144] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[145] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. *Current trends in Concurrency*, pages 510–584, 1986.

[146] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190. ACM, 1989.

[147] P.A. Porras and P.G. Neumann. EMERALD: Event monitoring enabling response to anomalous live disturbances. In *Proc. 20th national information systems security conference*, pages 353–365. NIST-NCSC, 1997.

[148] G. Proulx. Evacuation time and movement in apartment buildings. *Fire Safety Journal*, 24(3):229–246, 1995.

[149] G. Proulx. Evacuation by elevators: who goes first? In *Proc. Workshop on Use of Elevators in Fires and Other Emergencies*, pages 1–13. NRC Institute for Research in Construction, National Research Council Canada, 2004.

[150] L.L. Pullum and J.B. Dugan. Fault tree models for the analysis of complex computer-based systems. In *Reliability and Maintainability Symposium, 1996 Proceedings.'International Symposium on Product Quality and Integrity'., Annual*, pages 200–207. IEEE, 1996.

[151] J. G. Quintiere. Fire growth: an overview. *Fire technology*, 33(1):7–31, 1997.

[152] A. Rockstrom and R. Saracco. Sdl–ccitt specification and description language. *Communications, IEEE Transactions on*, 30(6):1310–1318, 1982.

[153] P. Rolin, L. Toutain, and S. Gombault. Network security probe. In *Proc. 2nd ACM Conference on Computer and communications security*, pages 229–240. ACM, 1994.

[154] R. Rosen, John J. Kineman Judith Rosen, and Mihai Nadin. *Anticipatory systems: philosophical, mathematical, and Methodological Foundations*. Springer, second edition, 2012.

[155] D.T. Ross. Structured analysis (sa): A language for communicating ideas. *Software Engineering, IEEE Transactions on*, (1):16–34, 1977.

[156] K. Scarfone and P. Mell. Guide to intrusion detection and prevention systems (IDPS). *NIST Special Publication*, 800(94), 2007.

[157] J. Schönefeld and DPF Möller. Runway incursion prevention systems: A review of runway incursion avoidance and alerting system approaches. *Progress in Aerospace Sciences*, 51:31–49, 2012.

[158] K. Sequeira and M. Zaki. ADMIT: anomaly-based data mining for intrusions. In *Proc. 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 386–395. ACM, 2002.

[159] F. Shang, S. Nara, T. Omi, Y. Goto, and J. Cheng. A prototype implementation of an anticipatory reasoning-reacting system. In *Proc. Computing Anticipatory Systems: CASYS - 7th International Conference, AIP Conference Proceedings*, volume 839, pages 401–414. AIP, 2006.

[160] GK Singh and C. Meier. Preventing runway incursions and conflicts. *Aerospace science and technology*, 8(7):653–670, 2004.

[161] S.E. Smaha. Haystack: An intrusion detection system. In *Proc. Fourth Aerospace Computer Security Applications Conference*, pages 37–44. IEEE, 1988.

[162] S.R. Snapp, S.E. Smaha, D.M. Teal, and T. Grance. The DIDS (distributed intrusion detection system) prototype. In *Proc. 3rd USENIX UNIX Security Symposium*, pages 227–233. USENIX, 1992.

[163] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An overview of ip flow-based intrusion detection. *Communications Surveys & Tutorials, IEEE*, 12(3):343–356, 2010.

[164] S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1-2):105–136, 2002.

[165] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids-a graph based intrusion detection system for large networks. In *Proc. 19th National Information Systems Security Conference*, pages 361–370. NIST-NCSC, 1996.

[166] W. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet detection based on network behavior. In *Botnet Detection*, pages 1–24. Springer, 2008.

[167] M.Y. Su, G.J. Yu, and C.Y. Lin. A real-time network intrusion detection system for large-scale attacks based on an incremental mining approach. *Computers & security*, 28(5):301–309, 2009.

[168] C.A. Sunshine, D.H. Thompson, R.W. Erickson, S.L. Gerhart, and D. Schwabe. Specification and verification of communication protocols in affirm using state transition models. *Software Engineering, IEEE Transactions on*, (5):460–489, 1982.

[169] A. S. Tanenbaum. Computer networks. *COMPUTER NETWORKS, ENGLEWOOD CLIFFS, PRENTICE HALL, US*, pages 141–148, 1996.

[170] H.S. Vaccaro and G.E. Liepins. Detection of anomalous computer session activity. In *Proc. 1989 IEEE Symposium on Security and Privacy*, pages 280–289. IEEE, 1989.

[171] A. Valdes. Detecting novel scans through pattern anomaly detection. In *Proc. 2003 DARPA Information Survivability Conference and Exposition*, volume 1, pages 140–151. IEEE, 2003.

[172] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. In *Proc. 3rd International Workshop on the Recent Advances in Intrusion Detection, Lecture Notes in Computer Science*, volume 1907, pages 80–93. Springer, 2000.

[173] G. Vigna and R.A. Kemmerer. NetSTAT: A network-based intrusion detection approach. In *Proc. 14th Annual Computer Security Applications Conference*, pages 25–34. IEEE, 1998.

[174] M. Von der Beeck. A comparison of statecharts variants. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 128–148. Springer, 1994.

[175] A. Wagner and B. Plattner. Entropy based worm and anomaly detection in fast ip networks. In *Proc. 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 172–177. IEEE, 2005.

[176] A.I. Wasserman. Extending state transition diagrams for the specification of human–computer interaction. *Software Engineering, IEEE Transactions on*, (8):699–713, 1985.

[177] G. White and V. Pooch. Cooperating security managers: Distributed intrusion detection systems. *Computers & Security*, 15(5):441–450, 1996.

[178] R. Wieringa. *Design methods for reactive systems: Yourdon, statemate, and the UML*. Morgan Kaufmann, 2003.

[179] J.R. Winkler and L.C. Landry. Intrusion and anomaly detection: ISOA update. In *Proc. 15th National Computer Security Conference*, pages 272–281. NCSC-CSL, 1992.

[180] W.A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, 1970.

[181] K. Yamanishi, J.I. Takeuchi, G. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Proc. 6th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 320–324. ACM, 2000.

[182] S.D. Young, D.R. Jones, United States. National Aeronautics, Space Administration, and Langley Research Center. Runway incursion prevention: a technology solution. In *Proc. Annual International Air Safety Seminar*, volume 54, pages 221–238. Flight Safety Foundation, 2001.

[183] P. Zave. A distributed alternative to finite-state-machine specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):10–36, 1985.

[184] Q. Zhao, J. Xu, and A. Kumar. Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation. *IEEE Journal on Selected Areas in Communications*, 24(10):1840–1852, 2006.

[185] Zurich Airport. 2011 runway safety report, 2011.

Index