

A Framework for an Adaptive Early Warning and Response System for Insider Privacy Breaches

PhD Thesis

Yasser M. Almajed

This thesis is submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

School of Computer Science and Informatics

De Montfort University

United Kingdom

February, 2015

Abstract

Organisations such as governments and healthcare bodies are increasingly responsible for managing large amounts of personal information, and the increasing complexity of modern information systems is causing growing concerns about the protection of these assets from insider threats. Insider threats are very difficult to handle, because the insiders have direct access to information and are trusted by their organisations. The nature of insider privacy breaches varies with the organisation's acceptable usage policy and the attributes of an insider. However, the level of risk that insiders pose depends on insider breach scenarios including their access patterns and contextual information, such as timing of access. Protection from insider threats is a newly emerging research area, and thus, only few approaches are available that systemise the continuous monitoring of dynamic insider usage characteristics and adaptation depending on the level of risk. The aim of this research is to develop a formal framework for an adaptive early warning and response system for insider privacy breaches within dynamic software systems. This framework will allow the specification of multiple policies at different risk levels, depending on event patterns, timing constraints, and the enforcement of adaptive response actions, to interrupt insider activity.

Our framework is based on Usage Control (UCON), a comprehensive model that controls previous, ongoing, and subsequent resource usage. We extend UCON to include interrupt policy decisions, in which multiple policy decisions can be expressed at different risk levels. In particular, interrupt policy decisions can be dynamically adapted upon the occurrence of an event or over time. We propose a computational model that represents the concurrent behaviour of an adaptive early warning and response system in the form of statechart. In addition, we propose a Privacy Breach Specification Language (PBSL) based on this computational model, in which event patterns, timing constraints, and the triggered early warning level are expressed in the form of policy rules. The main features of PBSL are its expressiveness, simplicity, practicality, and formal semantics. The formal semantics of the PBSL, together with a model of the mechanisms enforcing the policies, is given in an operational style.

Enforcement mechanisms, which are defined by the outcomes of the policy rules, influence the system state by mutually interacting between the policy rules and the system behaviour. We demonstrate the use of this PBSL with a case study from the e-government domain that includes some real-world insider breach scenarios. The formal framework utilises a tool that supports the animation of the enforcement and policy models. This tool also supports the model checking used to formally verify the safety and progress properties of the system over the policy and the enforcement specifications.

Declaration

The work described in this thesis is original work undertaken by the author for the degree of Doctor of Philosophy and no part of this material has been submitted for the award of any other qualification.

Yasser Almajed

Acknowledgements

In the name of Allah, the Entirely Merciful, the Especially Merciful. First and foremost, all praise to Allah who has given me the strength and bounty to complete this thesis.

This research work would not have been possible without the support of a number of people. I wish to express my gratitude to Dr. Helge Janicke for his expert guidance, constructive criticism, and critical comments. Also, I wish to thank Dr. Ali Al-Bayatti who was helpful and provided his invaluable assistance, understanding, and motivation. I also want to thank Dr. Antonio Cau for his technical suggestions and careful reading to improve this thesis. Additionally, I am grateful to Professor Hussein Zedan for the insightful discussion and support in the preparation of the research project. Also, my thanks go to past and present colleagues at the department for the friendly working atmosphere.

I would express my deepest thanks to my parents for their prayers, love, and encouragement throughout my studies.

Last but not least, I am greatly indebted to my wife and my children for their love, patience, and support over the years of PhD study.

Table of Contents

Abstract.....	I
Acknowledgements.....	IV
Table of Contents	V
List of Tables	IX
List of Figures.....	X
List of Abbreviations	XI
Chapter 1: Introduction	11
1.1 Introduction and Motivation	12
1.2 Problem Statement and Research Questions.....	15
1.3 Original Contributions	18
1.4 Research Methodology.....	19
1.5 Thesis Outline	21
Chapter 2: Background and Related Work	23
2.1 Introduction.....	24
2.2 Nature of Insider Breach Problem.....	24
2.2.1 Definitions of Insider and ‘Insiderness’	25
2.2.2 Categories of Insiders.....	28
2.2.3 Types of Insider Breaches.....	28
2.2.3.1 Insider IT Sabotage	29
2.2.3.2 Insider Theft of Intellectual property (Espionage)	29
2.2.3.3 Insider Fraud.....	29
2.2.3.4 Insider Privacy Breach	30
2.2.3.4.1 Categories of Insider Privacy Breach Scenarios	32
2.2.3.4.2 Facts and Cases about Insider Privacy Breaches	32
2.3 Existing Countermeasures against Insider Breaches.....	35
2.4 Usage Control	38
2.5 Insider Breach Problem and UCON.....	42
2.6 Policy Based Management.....	43
2.7 Summary	45

TABLE OF CONTENTS

Chapter 3: Adaptive Early Warning and Response System Architecture and Computational Model	46
3.1 Introduction	47
3.2 Architecture.....	48
3.2.1 Usage Decision.....	48
3.2.2 Interrupt Policy Decisions.....	51
3.2.2.1 Early warning levels	51
3.2.2.2 Interrupt Response Actions	53
Complex Response Actions	55
3.3 Computational Model	56
3.3.1 User model	60
3.3.2 AEWRC model	60
3.3.3 System model.....	61
3.4 Summary	62
Chapter 4: PBSL – Linguistic Support	63
4.1 Introduction	64
4.2 Privacy Breach Specification Language	65
4.2.1 Policy Rules	66
4.2.1.1 Event patterns	68
Basic event constructs	69
Derived Event Constructs	70
4.2.1.2 Attributes Constraints.....	71
4.2.1.3 Timing Constraints	73
4.2.1.4 Early Warning Actions	75
4.3 Some examples of PBSL usage	76
4.3.1 Example 1 (Data theft)	76
4.3.2 Example 2 (Masquerading)	78
4.3.3 Example 3 (Inference).....	79
4.3.4 Example 4 (Access for unauthorised purposes)	80
4.3.5 Example 5 (Access for a purpose without having a consent)	81
4.4 Summary	81
Chapter 5: PBSL – Formal Semantics	83
5.1 Introduction	84

TABLE OF CONTENTS

5.2	Labelled Transitions Systems (LTS).....	86
5.3	Fluent Linear Temporal Logic (FLTL)	86
5.4	Formal Semantics of PBSL.....	91
5.4.1	Event Patterns.....	91
5.4.2	Attributes Constraints	94
5.4.3	Timing constraints.....	95
5.4.4	Early Warning Actions.....	98
5.4.5	Policy Rules	99
5.5	Summary	101
Chapter 6: Enforcement.....		103
6.1	Introduction	104
6.2	Operational semantics of the enforcement mechanism.....	105
6.2.1	Policy Enforcement Behaviour	106
6.2.2	AEWRS System Behaviour	109
6.3	Efficient enforcement mechanism design	118
6.4	Summary	122
Chapter 7: Verification.....		123
7.1	Introduction	124
7.2	Case Study.....	124
7.2.1	Insider-aware Tax Revenue System (IATRS).....	125
7.2.2	Scenario Description	125
7.2.3	The informal system's acceptable usage requirements	126
7.2.4	Formalisation in PBSL.....	128
7.2.4.1	Very Low Early Warning Policy Rule	128
7.2.4.2	Low Early Warning Policy Rule	128
7.2.4.3	Medium Early Warning Policy Rule	129
7.2.4.4	High Early Warning Policy Rule.....	130
7.3	Verification	131
7.3.1	Safety Properties	132
7.3.2	Progress Properties.....	141
7.4	Summary	145

TABLE OF CONTENTS

Chapter 8: Conclusion	146
8.1 Summary of the Thesis.....	146
8.2 Contributions Revisited.....	149
8.3 Future Work	151
References	156
Glossary of Terms	168

List of Tables

Table 4.1 Syntax of <i>PBSL</i> : event Patterns	69
Table 4.2 Syntax of <i>PBSL</i> : Attributes Constraints	72
Table 4.3 Syntax of <i>PBSL</i> : Timing Constraints	74
Table 4.4 Syntax of <i>PBSL</i> : Early Warning Actions.....	76
Table 5.1 Syntax of <i>FLTL</i> : Logical operators	88
Table 5.2 Syntax of <i>FLTL</i> : Temporal operators	89
Table 5.3 Syntax of <i>FLTL</i> : Assertion and Ranges	89
Table 5.4 Formal Semantics of <i>PBSL</i> : Event patterns	92
Table 5.5 Formal Semantics of <i>PBSL</i> : Attribute Constraints	95
Table 5.6 Formal Semantics of <i>PBSL</i> : Timing Constraints: Interval.....	96
Table 5.7 Formal Semantics of <i>PBSL</i> : Timing Constraints: duration.....	97
Table 5.8 Formal Semantics of <i>PBSL</i> : Early Warning Actions.....	99

List of Figures

Figure 1.1 Security protection mechanisms timeline.....	14
Figure 1.2 Insider privacy breach scenarios on a time and risk level axes.....	17
Figure 2.1 TimeLine for the detection of the insider breach [98].....	33
Figure 2.2 Usage control actions [100]	39
Figure 2.3 Usage Control Model Components [76].....	40
Figure 3.1 Architecture of the extended UCON.....	50
Figure 3.2 Computational model	58
Figure 3.3 AEWRC	60
Figure 6.1 LTS representation for <i>REQ_TRIG_DelayWhenLowLevel</i>	107
Figure 6.2 LTS representation for <i>SuspendWhenMediumLevel</i>	108
Figure 6.3 LTS representation for <i>REQ_TRIG_AbortWhenHighLevel</i>	108
Figure 6.4 LTS representation for <i>REQ_TRIG_SkipWhenVeryLowLevel</i>	109
Figure 6.5 LTS representation for <i>DOM_PRE_POST_Suspended</i>	112
Figure 6.6 LTS representation for <i>DOM_PRE_POST_Delayed</i>	112
Figure 6.7 LTS representation for <i>DOM_PRE_POST_Aborted</i>	113
Figure 6.8 LTS representation for <i>DOM_PRE_POST_NormalOperation</i> ..	113
Figure 6.9 LTS representation for <i>REQ_TRIG_ResumeWhenDelayedAndTimeout</i>	115
Figure 6.10 LTS representation for <i>REQ_TRIG_ResumeWhenSuspendedAndAndEventOccurs</i>	116
Figure 6.11 LTS representation for <i>REQ_TRIG_AbortWhenSuspendedAndNoEventAndTimeout</i>	117
Figure 6.12 Concurrent and synchronous enforcement mechanism for conditions, clock synchronisation, and mutual exclusion	120

List of Abbreviations

AEWRC	Adaptive Early Warning and Response Controller
AEWRS	Adaptive Early Warning and Response System
E-Government	Electronic Government
ECA	Event-Condition-Action
FLTL	Fluent Linear Temporal Logic
LTS	Labelled Transition System
LTSA	Labelled Transition System Analyser
PBSL	Privacy Breach Specification Language
UCON	Usage Control

Chapter 1: Introduction

Objectives

- Explain the motivation for an adaptive early warning and response system
- Articulate the research questions
- Present the research contributions
- Provide the research methodology
- Outline the thesis structure

1.1 Introduction and Motivation

Data is the main asset of organisations including governments and healthcare bodies, and their information systems are used to manage personal information assets concerning individuals. These assets must be protected from outsider and insider threats. The protection of information assets against insider threats is of growing concern, because insider actions lead to more damaging consequences than those of outsiders [22]. One well-known case of an insider breach concerns a data leak from the UK Revenue and Customs department, in which the details of 25 million individuals (child benefit data) on two CD-ROMs were lost [14]. Hence, there is a need to protect organisations' information assets from unintentional and malicious insider threats.

The notion of a malicious insider threat has been defined as "...a current or former employee, contractor, or business partner who has or had authorized access to an organization's network, system, or data and intentionally exceeded or misused that access in a manner that negatively affected the confidentiality, integrity, or availability of the organization's information or information systems" [22]. Thus, data might be breached by insiders in different ways, depending on whether an insider: (1) has access to data; (2) has knowledge of data; and (3) is trusted by his organisation to access the data. This leads us to the notion of "insiderness" [17].

Insiderness, or the "degree of insiderness", was introduced in [17] to define the level of access a particular insider has with respect to a given asset, and measures the degree of access and knowledge of an asset, and the level of trust of the insider. This definition indicates that the insider may pose a different level of risk during the ongoing usage of information assets, depending on the above factors. Therefore, the information asset must be protected by an efficient security mechanism to counter insider threats taking into consideration the likelihood of occurrence of a breach to a digital information asset by a user at different risk levels of insider access.

Most current mechanisms present in the literature, such as conventional access control and auditing, only partially address the insider threats by reducing the potential insider threats, and limiting the opportunities for insider's breaches. Conventional access control mechanisms are used to limit inappropriate access of resources by determining whether to allow or deny prior to access being executed, but are unable alone to detect and prevent the insider breaches. This is because the insiders might misuse legitimate authorisations after the access has been granted. On the other hand, auditing mechanisms are used to log accesses to resources, and these log entries can only be analysed after access has been executed or even after the insider breach incident has been completed, thus rendering insufficient to interrupt insider activity (Figure 1.1). To the best of our knowledge, none of the existing mechanisms attempted to comprehensively counter such potential insider breaches.

Usage Control (UCON) has been proposed as a usage model which controls prior, ongoing, and subsequent usage of resources, encompassing the traditional access control, trust management, and data rights management [76]. The main features of this model are the mutability of attributes and ongoing control of long-running access. If attributes are changed during the ongoing usage, the access decision is either to continue access or revoke the user. Although UCON can be seen as a comprehensive model to reduce the odds for insider breaches, both, prior to and after access is granted, it does not support early warning and response process during the ongoing insider usage. The aim of this process is to effectively provide early detection; warn against insider breaches; delay, suspend, and interrupt the insider attacks in order to mitigate and counter insider breaches. Therefore, an early warning and response mechanism for insider breaches needs to be integrated into UCON within the comprehensive approach that is insider-aware in order to provide two layers of defence (Figure 1.1). This early warning and response system needs to take into consideration insider breach scenarios, including access patterns and timing constraints. In addition, to interrupt insider activity in dynamic software environments, where insider behaviour may dynamically change on the basis of the intent and context, there is a need to support multiple policy decisions at different risk levels, and adapt dynamically to new situations. We believe that it is important to specify a well-defined policy that explicitly defines insider breach scenarios.

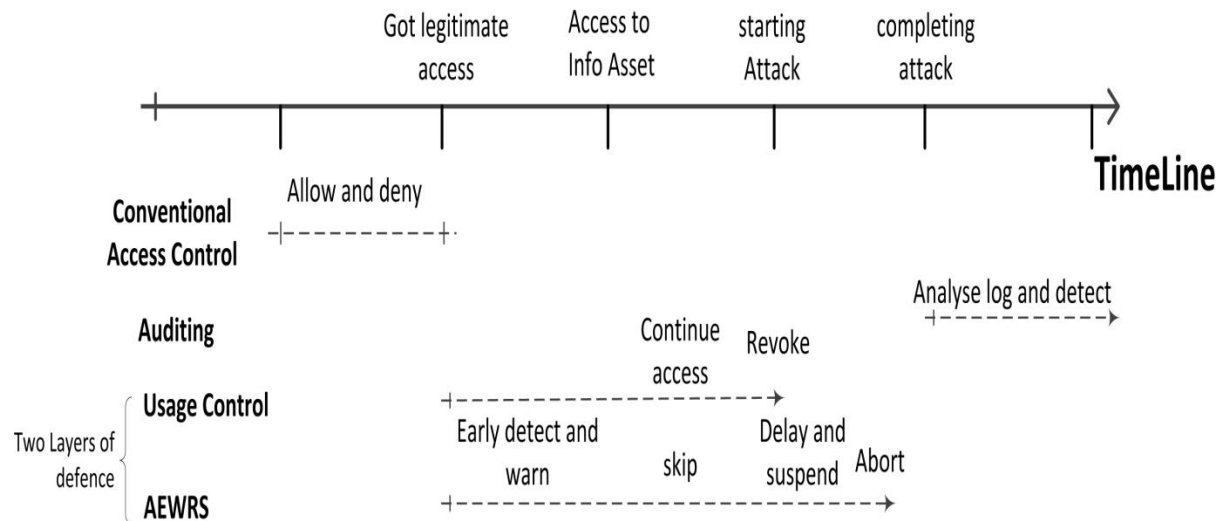


Figure 1.1 Security protection mechanisms timeline

Insider breach scenarios, business and security requirements, and legal regulations are continuously changing; therefore, policies to protect the assets need to be reviewed and adapted periodically to reflect the evolution of changes, and to deal with new situations of insider privacy breaches. Hence, we adopt a policy-based management approach [90] that allows policies to evolve dynamically to reflect system changes without the need to re-implement the system. The adaptability of this scalable approach is essential in large-scale, dynamic environment such as government or health bodies.

In this motivating example, we assume an operator within a government agency has legitimate access to personal information as part of his/her job. During working hours, the operator can access information stored in the organisation's database. In addition, we assume that there is a chance that insider privacy breaches (such as identity theft) may occur as a result of some social and psychological factors (disgruntlement or dissatisfaction) or financial problems. To conduct an insider privacy breach, the operator performs several steps: first, these particular data are accessed a couple of times within working hours, then the bulk data are queried outside of working hours, and finally the data are transferred to some removable media, printed, or emailed. We assume that the system has fundamental security mechanisms such as access control and auditing. The privacy breach scenario has multiple steps with

different levels of risk. If the access control mechanism detects the first step (browsing the data a couple of times) as a privacy breach, the system will deny or revoke the user. This is a strong decision that might be a false alarm—the full breach scenario has not yet occurred. In the case of auditing, a breach can only be detected after it happens, irrespective of the level of risk, so the system does not react in time, even in the case of actual data theft. It is not efficient for large, dynamic organisations to rely on human intervention to check numerous log entries. Thus, there is a crucial need for a system that enables early warnings and can enforce adaptive and proactive actions (such as suspend and delay access) when suspicious events are detected. This would mitigate and warn against risks, prevent insider activities before the actual attack takes place, and adapt dynamically to new situations. All of this should be achieved during real-time system use, depending on the insider's access history, and within certain timing constraints.

1.2 Problem Statement and Research Questions

Protection from insider threats is a nascent research area in the field of information security. Insider threats are a very complex problem to handle within dynamic software systems, because the insiders have legitimate access to information assets and are trusted by their organisations. The nature of insider privacy breaches changes depending on the organisation's acceptable usage policy and the attributes of an insider: access, knowledge, and trust. However, the level of risk that insiders pose (i.e. severity level of insider access) depends on insider breach scenarios including their access patterns and contextual information, such as timing of access.

Few approaches systemise the continuous monitoring of dynamic insider usage characteristics and adaptation. Most of current approaches present in the literature attempt to solve particular problems of insider threats, and are focused only on mechanisms of prevention before the access is executed, e.g. traditional access control systems, or after the access is completed, e.g. auditing systems. UCON, although, a comprehensive model that is useful in handling insider threat problem, supporting the continuous monitoring during the ongoing usage after access has been granted, it has not completely solved the insider problem, especially in a dynamic software system environment.

CHAPTER 1. INTRODUCTION

Designing an adaptive mechanism to address insider breaches which is compatible with a dynamic software system environment is a very challenging task. The system must be able to adapt its behaviour based on ever-changing insider scenarios. Thus, it must respond to changes in the severity level of insider access according to different scenarios of insider breaches if it is to enforce adaptive response actions. Therefore, a security system that is utilised in a dynamic software system environment in order to enforce adaptive response actions must be policy-based.

Policies must be adapted to, in a dynamic manner, on the basis of changes in the system state. Thus, the policy decisions must depend on the current and past insider behaviour. Therefore, a policy-based system that is utilised in a dynamic software system environment must be dynamic, in which policies can be adapted dynamically in response to the occurrence of events or depending on time.

To the best of our knowledge, none of the existing mechanisms attempt to comprehensively counter the potential insider breaches, where the insiderness is a vital aspect of any dynamic software system, because the risk levels of insider access may dynamically change on the basis of their access patterns and timing constraints (Figure 1.2). Thus, it is essential that an adaptive early warning and response system that is suited for a dynamic software system environment is developed in order to rectify the loopholes of existing security mechanisms.

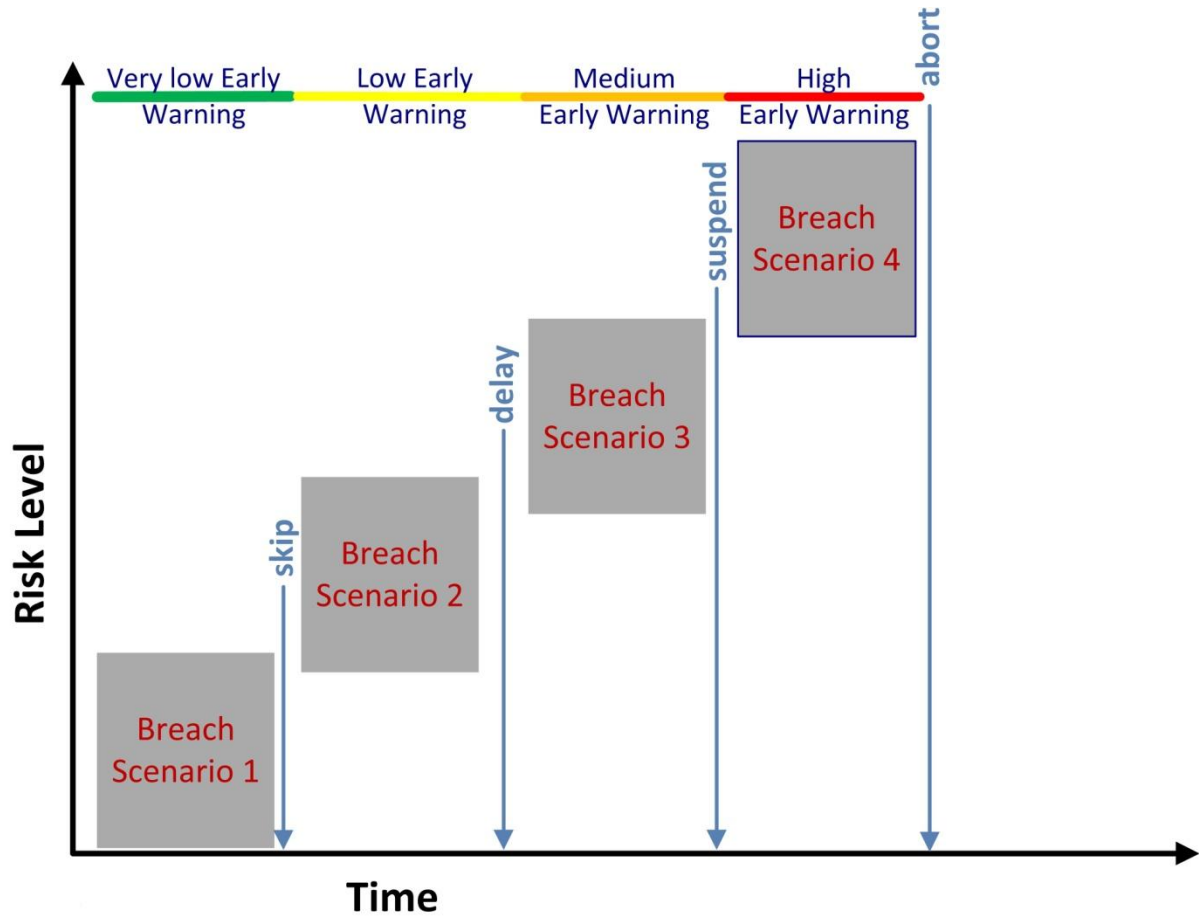


Figure 1.2 Insider privacy breach scenarios on a time and risk level axes.

In this thesis, we propose a formal framework for an adaptive early warning and response system for insider privacy breaches within dynamic software systems. This framework allows for the specification of multiple and dynamic policies at different risk levels during usage (depending on insider breach scenarios), and for enforcement of adaptive response actions to interrupt the insider activity. The main features of our framework are: (1) the protection from insider breaches in a timely fashion through multiple interrupt actions, (2) the continuous monitoring of insider normal operation enabled with early warning provisions, (3) a high probability of early detection of insider breaches without an unacceptable number of false alarms by enforcing delay and suspend access, and the ability to express comprehensive insider breach scenarios, and (4) a minimum of human intervention through the adaptation process.

CHAPTER 1. INTRODUCTION

The key research question is:

How can we develop a framework for an adaptive early warning and response system for insider privacy breaches within a dynamic software system environment?

To answer the research question, six sub-questions can be formulated:

- 1- What is the nature of the insider breach problem, and what existing approaches protect from insider breaches in the software system environment?
- 2- How can adaptive early warning and response processes be incorporated into usage control model?
- 3- How can an expressive policy specification language be developed that captures the access patterns and timing constraints?
- 4- How can the policy specification language and the mechanism of enforcing the policies be suitably formalised so that policies can be enforced on the system?
- 5- How can we evaluate the expressiveness of the policy language on a case study from the e-government domain that includes some real-world insider breach scenarios?
- 6- How can verification techniques such as model checking be used to verify the policy and the enforcement mechanism model against the properties of the system?

1.3 Original Contributions

The contributions of this thesis can be summarised as follows:

- 1- The UCON model is extended to include interrupt policy decisions, in which multiple policy decisions can be expressed at different risk levels during (ongoing) usage (**Chapter 3**).

CHAPTER 1. INTRODUCTION

- 2- We develop a computational model to demonstrate the abstract concurrent behaviour of an adaptive early warning and response system for a real information system in the context of policy-based management (**Chapter 3**).
- 3- We introduce the Privacy Breach Specification language (PBSL) based on the computational model (**Chapter 4**).
- 4- The policy language and the mechanism enforcing the policies are formalised in an operational style, allowing policies to be enforced on the system (**Chapter 5 - 6**).
- 5- PBSL is evaluated using a case study from the e-government domain (**Chapter 7**).
- 6- The safety and progress properties of the policy and enforcement mechanism models are formally verified using an automatic verification tool (**Chapter 7**).

1.4 Research Methodology

Our research addresses a software engineering problem using a constructive research method. This method constructs or develops novel artefacts, such as a framework, language, theory, and algorithm, to solve the research problem.

The methodology of the proposed approach is comprised of four work packages. The first address the research background and related work, and the second focuses on the proposed model and language. The third work package presents suitable formal semantics for the language and the model. The final work package concentrates on evaluating our work.

Work package 1: Research background and related work

The research background and relevant literature are assessed, utilising digital resources and books. We consider aspects related to the nature of the insider breach

problem, and existing mechanisms to protect from insider threats such as access control, anomaly detection, and monitoring mechanisms (**Chapter 2**). This enables us to articulate the key research question and identify how the research problems can be understood and solved.

Work package 2: Model and language

To determine how we can answer the key research question, we propose a computational model of the system and the PBSL (**Chapter 3 - 4**). The research in this work package explores the link between the policy language and the computational model in the context of policy-based management. This package captures the second and the third research sub-questions.

Work package 3: Formal Semantics

Based on the computational model and policy language derived in the previous work package, this package explores existing formal methods in order to develop semantics for the policy language and model. Thus, the selected formal method is utilised to give the policy and enforcement mechanism operational semantics (**Chapter 5 - 6**). This package captures the fourth research sub-question.

Work package 4: Evaluation

To demonstrate the validity of our approach, we use a case study from the e-government domain that includes some real-world insider breach scenarios (**Chapter 7**). The automatic verification tool is used to formally verify the system properties over the policy and the enforcement mechanisms models (**Chapter 7**). This package captures the fifth and sixth research sub-questions.

1.5 Thesis Outline

This thesis is structured into eight chapters:

Chapter 1: Introduction

This provides the motivation for an adaptive early warning and response system, and articulates the research questions and objectives.

Chapter 2: Background and Related Work

This chapter defines an insider, the notion of insiderness, and various types of insider threats, including insider privacy breaches. We then review some countermeasures to protect from insider threats, such as access control and anomaly detection and monitoring mechanisms. The UCON model is then introduced, and the adoption of this model to deal with insider breaches is investigated. Finally, an overview of policy-based management is given, reviewing some existing policy models.

Chapter 3: Adaptive Early Warning and Response System Architecture and the Computational Model

This chapter presents the architecture of our proposed adaptive early warning and response system (AEWRS) for insider privacy breaches, which extends the traditional UCON model. We then propose a computational model for AEWRS. The model demonstrates the abstract system components and their behaviour and interactions in the context of policy-based management.

Chapter 4: PBSL – Linguistic Support

This chapter introduces the PBSL, based on the computational model in Chapter 3. The syntax of the language is presented and described, privacy breach scenarios including the events patterns and timing constraints that determine a certain early warning action are specified in a unified manner in the form of policy rules. Finally, some examples of insider privacy breach scenarios are given in the form of PBSL specifications to demonstrate the usage of the language.

Chapter 5: PBSL – Formal Semantics

Chapter 5 outlines the formal semantics of PBSL components in labelled transition systems (LTS) and fluent linear temporal logic (FLTL). Graphical representations of PBSL components are presented in LTS, which is based on state machines, computed from asynchronous FLTL specifications using a labelled transition system analyser (LTSA).

Chapter 6: Enforcement

In this chapter, the formal operational semantics for the enforcement mechanism of the AEWRS system is given in LTS and FLTL using LTSA. An efficient enforcement mechanism design is proposed.

Chapter 7: Verification

This chapter presents a case study of tax revenue systems as a means of evaluating our approach. A formal verification of the safety and progress properties is described using the LTSA model checker.

Chapter 8: Conclusion and Future Work

This chapter summarises our research and its findings, and provides suggestions for future work.

Chapter 2: Background and Related Work

Objectives

- Define the nature of the insider threat problem and identify various types of insider threats, including insider privacy breaches.
- Review related work in the area of protection from insider threats.
- Overview of the Usage Control Model (UCON).
- Investigate how Usage Control Model (UCON) can be adopted to deal with insider breaches.
- Explore the policy-based management approach and review the related policy models.

2.1 Introduction

In this chapter, we provide background information on the nature of the insider breach problem. We begin with the notion of an insider, presenting different definitions of this term and types of insiders and follow with an introduction of the notion of ‘insiderness’. Types of insider breaches are described, including insider privacy breaches. Some existing approaches in the area of protection from insider threats including anomaly detection and access control are critically reviewed. In addition, we illustrate the usage control model (UCON) and describe the differences between some existing formal approaches in order to specify and analyse this model as well as their limitations. We investigate how the UCON model can be employed to address the insider breach problem. Next, an overview of the policy-based management approach is discussed, reviewing some existing policy models, dynamic policies, and the importance of using the formal semantics of policy languages for security-critical and dynamic environments, such as e-government.

2.2 Nature of Insider Breach Problem

Data protection is one of the major concerns in many large-scale, dynamic software system environments, such as e-government, financial, and healthcare systems. Organisations have reasonability to collect information from individuals in order to facilitate services for them. The protection of digital information assets against insider threats is a major concern for large enterprises where their digital repositories are used to store personal information assets concerning people. The software systems are vulnerable to insider privacy breaches because insiders have direct authorised access to individuals’ information assets; and are trusted by the organisation to access them.

The nature of the insider privacy problem, where an insider compromises information assets, changes depending on the organisation’s acceptable use policy and data protection regulations. The organisations are responsible for the privacy and confidentiality of the assets that they manage, ensuring that they are not used by their employees, who have authorised access for unauthorised purposes.

Insiders may have different motivations, such as personal, financial, and psychological factors. Insiders could act individually or in collusion with others. They could be malicious (intentional) or non-malicious (unintentional). Insiders may have some level of access to some or all information assets, have authority over system operations, or have knowledge of sensitive personal information. In addition, they may have technical skills and experience. Hence, they may have opportunity in a position of trust to commit a malicious act during normal operating conditions [45].

These breaches could lead to a high impact on agencies as well as on individuals, including financial loss, disruption to the organisation, loss of customer trust in organisational services, and legal actions against the organisation [18]. On the other hand, loss of privacy has a high impact on national security as well.

2.2.1 Definitions of Insider and ‘Insiderness’

It is important to define an insider in an unambiguous way in order to address insider breach issues. Defining “insider” could be very difficult, as insider is a new and complex concept. Recently, this term has begun to be used and understood in different ways, but there is no general agreement on a standard definition. There are several definitions of insider in literature, and they vary depending on different perspectives. This has recently been discussed by a wide range of researchers.

The term ‘insider’ has been defined from the perspective of access, knowledge, and trust that an insider has, as follows:

- 1- A number of workshops by RAND that started in 1999 attempted to define the term ‘insider’ [4], [5], [19]. They defined insider as “*someone with access, privilege, or knowledge of information systems and services*”. This definition includes the access and knowledge factors but does not include that the insider is implicitly trusted.

CHAPTER 2. LITERATURE REVIEW

- 2- A similar definition of insider by Greitzer et al. [39] is “*an individual currently or at one time authorised to access an organisation’s information system, data, or network*”.
- 3- A broad definition of an insider in terms of trust that includes information and non-information assets has been described by Bishop et al. in [18] as “*a person that has been legitimately empowered with the right to access, represent, or decide about one or more assets of the organisation’s structure*”. That implies both logical and physical presence.

Some researchers distinguish between logical and physical insiders in that the insider may be inside or outside the system perimeter. Logical insiders may be physically outside the system perimeter, such as a user on duty accessing the system remotely, or physical insiders may be logically outside, such as a caretaker not actually having access to the system. Examples of these definitions that distinguish in terms of “insider” and “outsider” from the perspective of the system perimeter are as follows:

- 1- “*Anyone operating inside the security perimeter*” as cited in [18].
- 2- “*Person who is allowed inside the security perimeter of a system and consequently has some privileges not granted outsiders*” as cited in [73].

These definitions lack the clear distinction of the security perimeter with the assumption that there is only one security perimeter.

Others describe “insider” in terms of different levels of access to a given information asset rather than distinguish in terms of “insider” and “outsider”. In other words, attackers have degrees of access with respect to an asset leading to “degrees of insiderness” as introduced by Bishop et al. [17]. This definition indicates that the access may pose a different level of risk during the ongoing usage of information assets regardless of the attacker being inside or outside the security perimeter.

The notion of insiderness deals with aspects regarding an insider that include having access to and knowledge of a resource by one who is considered trustworthy by an organisation, which may pose different levels of risk against that resource depending on those factors. This is different from regarding insider threats as an “insider” or “outsider”, as most security mechanisms are designed on that basis. These mechanisms, such as access control, normally define binary policy decisions (namely allow and deny), apply them statically, and use strong enforcement (e.g., deny or revoke). The ability to delineate multiple possible policy decisions at different risk levels rather than just natural access control binary ones and apply them dynamically, rather than statically provides proactive decisions before enforcing reactive decisions and helps to adopt insiderness [17]. We find that this definition is the most appropriate definition for our work, as it takes into consideration the likelihood of occurrence of a breach to a digital information asset by a user at different risk levels of insider access.

“Insider threats” have been defined from the perspective of intent, motivation, and the impact of the threat:

- 1- The notion of a malicious insider threat has been broadly defined by the CERT program of Carnegie Mellon University’s software engineering institute as “*a current or former employee, contractor, or business partner who has or had authorized access to an organization’s network, system, or data and intentionally exceeded or misused that access in a manner that negatively affected the confidentiality, integrity, or availability of the organization’s information or information systems*” [22]. This definition is IT-specific and considers only malicious (intentional) insider threats without acknowledging accidental (unintentional) ones. From the organisation’s perspective, an insider can be an employee, ex-employee, temporary employee, contractor, business partner, temporary business partner, or some other individual linked to IT.
- 2- RAND describes insider threats as “*malevolent (or possibly inadvertent) actions by an already trusted person with access to sensitive information and information systems*” [19]. This considers both intentional and unintentional insider threats.

- 3- Greitzer et al. define insider threats as [39], “*harmful acts that trusted insiders might carry out; for example, something that causes harm to the organisation, or an unauthorised act that benefits the individual*”. This considers only intentional actions, the motivation for those actions, and their consequential risk to the organisation.
- 4- The Centre for the Protection of National Infrastructure (CPNI) defines insider [threat] as “*a person who exploits, or has the intention to exploit, their legitimate access to an organisation’s assets for unauthorised purposes*” [23]. This definition is more general than the previous ones and includes both information and non-information assets and considers both intentional and unintentional insider threats.

2.2.2 Categories of Insiders

Insider threats have been defined by differentiating between three types of insiders [83], [85] based on legitimate access, knowledge, and intent as follows:

- 1- Masquerader: an individual who uses legitimate insider credentials to impersonate the legitimate user for malicious purposes.
- 2- Traitor: a legitimate insider who uses his/her own legitimate credentials to violate his/her origination’s security (usage) policy maliciously.
- 3- Naive insider: a legitimate insider who uses his/her own legitimate credentials to violate his/her origination’s security (usage) policy unintentionally.

2.2.3 Types of Insider Breaches

The classifications of insider breaches have been derived from surveys conducted, such as by CERT. The CERT program at Carnegie Mellon University in the US carried out an in-depth analysis of over 700 case studies of insider crimes since 2001. The following types are based on the perspective of motivation and potential impact of the risk.

2.2.3.1 Insider IT Sabotage

“Insider use of information technology to direct harm at an organisation or an individual” [22]. The majority of those insiders were privileged insiders with technical skills and access to the organisation’s IT facilities. Most of those insiders had some disturbing attributes, such as conflicts with colleagues or psychological-social factors. The motivating factor includes the desire to gain revenge due to disgruntlement.

Examples of sabotage include:

- Planting a logical bomb to destroy critical data.
- Deleting critical data, such as data copies and backup files.
- Disturbing critical system operations.

2.2.3.2 Insider Theft of Intellectual property (Espionage)

“Insider use of IT to steal proprietary information from the organisation, this category includes industrial espionage involving insiders” [22]. The majority of the insiders that stole intellectual property (IP) were scientists, engineers, or programmers. According to CPNI and CERT, they were driven to take them to a new competing employer, to form their own business, or to sell the trade secrets to a competitor.

Examples of stolen IP assets include the following:

- Proprietary software and source code.
- Business plans.
- Confidential product information, such as engineering designs and scientific formulas.

2.2.3.3 Insider Fraud

“Insider use of IT for the unauthorized modification, addition, or deletion of an organization’s data (not programs or systems) for personal gain, or the theft of information that leads to an identity crime (identity theft, credit card fraud)” [22]. Most of the insiders

CHAPTER 2. LITERATURE REVIEW

who commit fraud are low-level employees, such as data entry clerks or those in administrative positions. Unlike the insiders of IT sabotage and IP theft, they are without technical skills and usually not professionals. They are mostly motivated by financial gain.

Examples of insider fraud include:

- Making a false payment for financial benefits.
- Making a false payment to be redirected to an account not that is not the customer's account.
- Making a purchase order and approval by the same user.

2.2.3.4 Insider Privacy Breach

There are different definitions of a privacy breach in literature that are appropriate for particular circumstances, study disciplines, and organisation or application domains, yet there is no definition that is suitable for all these domains. According to our scope in this thesis, it is essential to have a basic definition of an insider privacy breach to be a useful starting point for proposing the appropriate technical approach to deal with it. Hence, we define an insider privacy breach as follows:

Insider use of IT for the access of personal data within an organisation in an unacceptable way for personal gain, or the theft of information that leads to an identity crime.

In order to differentiate between an insider privacy breach and insider fraud, we consider here insiders who access personal data for their own interest and not those who add or modify only to commit financial fraud. In addition, insider privacy breaches in the definition include both data accesses and theft of personal information. The unacceptable way in our definition means that the way in which insider accesses data is inappropriate that may result in privacy breach and data theft. Although the insider is expected to be entrusted with access (e.g. has authorised access as a part of his/her job), the behaviour is unacceptable, leading to serious incidents at higher levels of risk. In the context of security policy, the data is assumed to be accessed according to the expected behaviour, and the authorised user is

CHAPTER 2. LITERATURE REVIEW

expected to be trusted not to breach the security policy. In the context of the acceptable use policy, the insider has already been granted authorised access to data but she/he behaves in a way that violates the acceptable use policy that he/she is intended to adhere to. Considering whether the use is acceptable or unacceptable normally depends on the organisation's acceptable usage policy. As in insider fraud, most of the insiders are low-level employees and could act individually or in collaboration with others. They commit privacy breaches for personal interests or financial factors.

Privacy protection has been largely recognised from the legal perspective. There are a number of legal regulations to protect data privacy, such as the Data Protection Act of 1998 [75] in the UK, the Data Protection Directive 95/46/EC [32] in Europe, the Health Insurance Portability and Accountability Act (HIPAA) [2] in the US, and the Personal Information Protection and Electronic Documents Act [80] in Canada.

According Directive 95/46/EC (article 2), personal data has been defined as [32]: *“Any information relating to an identified or identifiable natural person (“data subject”); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity”*. This definition is very broad, including the information that allows direct or indirect identification of the data subject (i.e. the person to whom personal data belong). Additionally, it involves sensitive personal data such as those related to race, health, religious beliefs and political opinions [32] (article 8).

2.2.3.4.1 Categories of Insider Privacy Breach Scenarios

Based on the above definition of an insider privacy breach, insider privacy breaches are classified into the following:

- 1- Inappropriate access to particular personal data.

Examples of this category include:

- Individual snooping—deliberate browsing where details of information have been accessed through direct access in an unacceptable way (e.g., for unethical purposes, without consent, or not related to the insider's duties).
- Access of unauthorised data through indirect access (e.g., data inference) [84].
- Multiple anomalous accesses to personal data.
- Masquerading by using a legitimate insider's credentials to access the personal data.

- 2- Anomalous or inappropriate access to a large amount of personal data.

- 3- Leak of a massive amount of personal data through either a malicious data theft (e.g., identity theft) or an accidental data loss [65].

2.2.3.4.2 Facts and Cases about Insider Privacy Breaches

According to CPNI, the Insider Data Collection Study [23] collected and analysed data on 120 insider cases in the UK between 2007 and 2012. The majority of insider activities identified were unauthorised disclosure of sensitive information (47%) and insider fraud (42%).

The Information Security Breaches Survey 2014 [95] revealed that 58% of large organisations have experienced insider (namely staff) security breaches. Most of those incidents involved unauthorised access to systems or data, such as masquerading. This

CHAPTER 2. LITERATURE REVIEW

affected 57% of large organisations that have had security breaches from their staff. The data indicates that the leakage of confidential information is the second most reported breach type in more than half of large organisations at 55% compared to a third (33%) that experienced security incidents caused by the misuse of confidential data. The survey indicates that 45% of those organisations experienced data protection breaches. This type of breach resulted in effects on those organisations, including large regulatory fines, influences on brand reputation, and high costs in terms of investigation and resolution.

The Information Security Breaches Survey 2014 [95] revealed some financial costs to fix the worst security incidents of that year in large organisations. The average time to repair the breaches was high with an average of 45 to 85 days. Large organisations incurred £12,000 to £34,000 in time costs and £80,000 to £135,000 in cash costs on average as a result of those breaches.

Verizon 2014 Data Breach Investigations Report [98] consists of 1017 incidents concerning the timeline data for the detection of a breach, Figure 2.1 illustrates the timeline for the discovery of a breach. There is a mere 13% of incidents that took minutes or less to detect. The majority of the breach incidents were discovered within days or longer (69%), which is very significant.

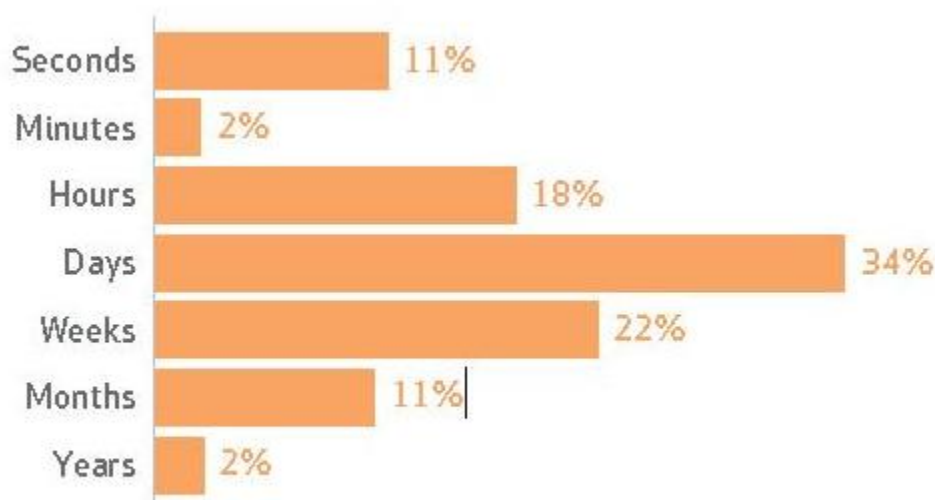


Figure 2.1 TimeLine for the detection of the insider breach [98]

CHAPTER 2. LITERATURE REVIEW

The UK Audit commission in “Ghost in the machine an analysis of IT fraud and abuse” [10] reported a case of insider privacy breach. A nurse on night duty at a hospital was using her authorised access to search for names of friends and family on the patient administration system that were not linked to her official duty. She then discussed the health problems of the individuals that she searched for with other members of her family. This was considered a breach of confidentiality, and the nurse was given a written warning.

Several realistic case studies have been presented in the report “Insider Threat Study: Illicit Cyber Activity in the Government Sector” conducted by CERT and the US Department of Homeland Security [53]. One of those cases involved a project leader who took a new position in a different department within the same organisation. He convinced supervisors in his former department to allow him to keep an account on their system because it was a mutually agreeable solution. That access, although with lower access rights than before, combined with his knowledge of additional access methods on the network, enabled him to repeatedly increase access rights on his account, even as the increased access was discovered and corrected by the system administrator. Using the elevated access, he logged in after hours and accessed confidential personnel and payroll files.

Several realistic case studies have been reported in “2013 Information Security Breaches Survey-Technical Report” [94] commissioned by the Department for Business, Innovation, and Skills in the UK. They revealed “a disgruntled employee at a large utility company stole some sensitive information which he had access to as a part of his job and began selling this. The breach was discovered by accident, over a month after it started. The value of the lost data was several hundred thousand pounds, but the impact on the business of the investigation and aftermath was even greater. The lack of a contingency plan contributed to this cost. After the breach, the company deployed new systems, changed its procedures and introduced a formalised post-incident review process.”

In this report “2014 Information Security Breaches Survey - Technical Report”, they reported “an employee of a social care facility based in the South East of England repeatedly gained unauthorised access to confidential information about individuals he knew in order to

facilitate fraudulent activities. As a result, the police were informed and disciplinary steps were taken against the employee. Stricter staff training and vetting process were also implemented as a result” [95].

2.3 Existing Countermeasures against Insider Breaches

Most current mechanisms in literature, such as conventional access control and auditing, incompletely address the insider breach problem by reducing the potential for insider threats and limiting opportunities for insider breaches. Access control mechanisms are used to limit inappropriate access of assets by determining access control decisions, namely either to allow or deny prior to access being executed, but alone are unable to detect and prevent insider breaches. This is because insiders might misuse legitimate authorisations after access has been granted. Auditing mechanisms are used to log access of resources, and the analysis of those log entries can be only performed after access has been executed or even after the insider breach incident has been completed. Hence, it is insufficient to interrupt insider activity as the attacks are detected after insiders have breached data. Therefore, there is a need to prevent the attack before it happens by allowing early warning.

Early warning is one of the main components of intrusion prevention systems—the new generation of intrusion detection systems. Early warning has received little attention because they can increase false positives. Early warning techniques, such as sandboxing and honeypots [91], [92], are designed mostly to detect outsider attacks and have some limitations. They require human intervention and do not support dynamic strategies for real-time responses to detected misuse.

Most of the proposed misuse specification languages have been built on misuse detection systems. Misuse by insiders of data cannot be specified by most existing misuse specification languages, such as *STATL* [33], *CARDS* [99], *Sutekh* [81], *P-BEST* [61], *RUSSEL* [72], *LAMBDA* [28], *ADELE* [76], *IDIOT* [55] and *SHEDEL* [66]. This is because they are designed specifically to specify intrusions of outsiders and not to specify misuse by insiders.

CHAPTER 2. LITERATURE REVIEW

It could be argued that some responses of outsiders and insiders could be similar [73], in some cases, and some specification languages allow response actions to partial misuse scenarios (i.e., those that have not been completed yet, such as to raise a yellow alert), block the user, or alert the system administrator. In the case of blocking access, the insider breach scenario might be petty or have multiple steps but has not been completed (i.e., no further steps of the attack has happened after the initial step), which could be a false alarm. While in the case of an alert or log, it is costly and impractical in a large dynamic organisation to use only human intervention to check a high number of alerts and also to monitor any further suspicious events.

Automatic and proactive response has received little attention in literature in intrusion detection. That is probably due to the inherent complexity and risk in developing automatic responses, which could lead to a disturbance to legitimate users and a warning to intruders. Neumann [73] argues that insider misuse should be detected and allowed to continue with online surveillance. To confine the effects of serious attacks, dynamic strategies for real-time response to the detected misuse and anomalies must be adopted.

Schutzer [88] recommends that monitoring tools in the financial services industry should be developed with forecasting capabilities to forecast an employee's intention before they commit an identity crime. These include the likelihood of the threat based on factors, such as employee behaviour that fits a known criminal pattern. Also, these tools should have the ability to take response actions, such as timely reporting.

A privacy intrusion detection system is introduced by Venter et al, [97], and sets of features have been defined to detect possible insider privacy breaches based on the anomalous behaviour of insiders. They suggested that once the insider exceeds the threshold of one feature, such as frequency of access to a particular record, the system would react accordingly to block access, alert, or reduce the threshold of the feature (called throttling). They employ anomaly-based detection and provide only conceptual architecture for their proposed system without testing. It is clear that this approach can cause many false alarms, as the system checks each feature one-by-one, irrespective of the full scenario of the insider.

CHAPTER 2. LITERATURE REVIEW

That means it does not have the ability to predict the likelihood of a potential privacy breach that is happening or is going to happen.

An et al. [3] attempted to address the problem of the uncertainty of privacy intrusion by evaluating the probabilities of all the features together using Dynamic Bayesian Networks. However, the approach was to detect a large amount of data theft regardless of the user, and it is not effective for detecting privacy breaches for a particular individual or group of individuals, as they claim. That causes false negatives, while true intrusions were not detected.

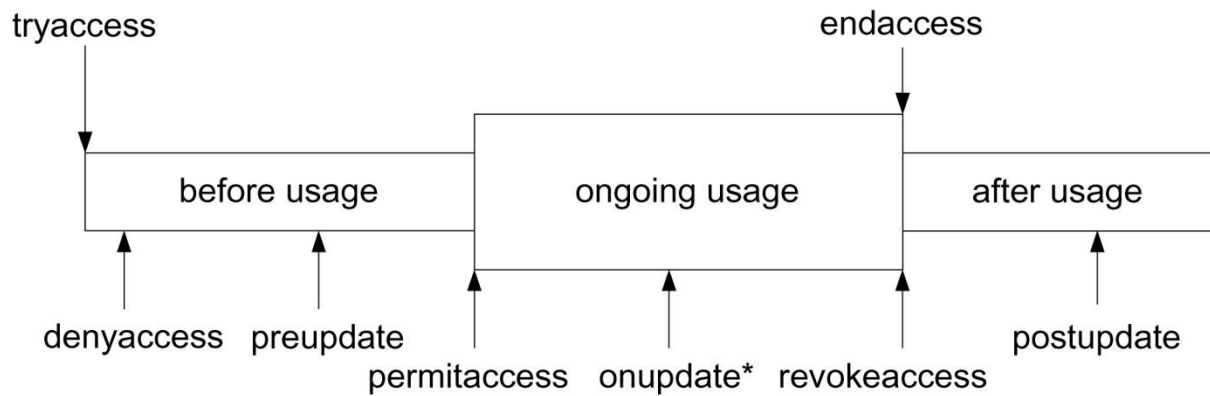
Kamra and Bertino [51] proposed a framework for an anomaly detection system for a database management system (DBMS). They proposed what they call a privilege state based access control mechanism for supporting multiple response actions to anomalies. This uses a response engine based on a policy language. The language is based on event-condition-action (ECA). The disadvantage of the language is that it is not expressive enough to represent more complex insider breach scenarios, including event patterns. Furthermore, it does not support timing constraints.

A conceptual framework is proposed by Crampton and Huth [25]; they defined a general-purpose policy language and an extended access control framework to deal with insider threats. Their language does not specify how the full breach scenario with multiple steps is defined, and it is not rich enough in terms of notations and operators to express complex insider breaches, including access patterns and timing constraints. They only used abstract requests and context predicates with limited operators. Additionally, their approach does not support early warning and multiple policy decisions. We argue that an approach should be integrated with expressive breach specification language (including the basic sorts of operators and timing constraints) to specify the known patterns of breach scenarios in order to clearly realise early warning and to avoid false alarms. Unlike our approach, they did not use underlying formal semantics to formalise the language and analyse the system properties. Additionally, they do not evaluate their approach on existing case studies.

Probst and Hansen [82] proposed an abstract model in terms of insider capabilities and restrictions to deal with the insider problem, providing formal semantics for their model using process algebra. They present a modelling language without providing the formal semantics of the language. They use forensic static analysis to analyse the access control specifications after the attack, the same as the auditing mechanism, where data is analysed after the fact. They use over-approximation for their analysis to detect insider attacks with the aim of lowering false positives. However, they deal mainly with physical systems based on the notions of locations and actors to check whether a location is at risk of the insider treating the location and data as resources. Additionally, they did not use the notion of time and frequency of access to the data asset to assess a data asset is at risk of an insider at run-time.

2.4 Usage Control

Usage Control (UCON) is a usage model that encompasses traditional access control, including Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-based Access Control (RBAC), trust management, and data rights management [76]. The main features of this model are the notions of mutability of attributes [77] and continuity of decisions. The mutability means that attributes associated with subject and object can be updated during ongoing usage as a side effect of the subject's access to the object, which influences the system state. Examples of these mutable attributes are the context of the user including time or location. Continuity of decisions means that control is performed prior (pre), ongoing (on), and subsequent to (post) the usage of resources. If attributes are updated during ongoing usage, the access decision is either to continue access or revoke the user according to the policy. During a usage process life cycle, there are two categories of usage control actions: subject actions and system actions [100] (Figure 2.2). Subject actions are those performed by a subject including *tryaccess* and *endaccess*, while the system actions are those performed by the system including *permitaccess*, *denyaccess*, *revokeaccess*, as well as the update actions that update the value of an attribute: *preupdate*, *onupdate* and *postupdate*. The system states change by updating the mutable attributes as side effects of subject actions. Therefore, UCON can be seen as a comprehensive model to not only control resources before access is granted, but also after it has been granted when the resource is being accessed.

Subject Actions**System Actions****Figure 2.2** Usage control actions [100]

The usage process can be viewed as long-running access. To determine whether to allow or deny access rights, $UCON_{ABC}$ introduced by Park and Sandhu in [76] depends on authorisation, obligation, and condition components (Figure 2.3). It incorporates conventional access control models, including conventional ones, such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-based Access Control (RBAC) as well as some other existing ones. Authorisation is a key functional requirement that must be fulfilled before allowing a particular right of access to a digital object. Authorisation must be evaluated both before and during access. Obligation is used to confirm the mandatory requirements that a subject must undertake both before and during a particular usage process. These mandatory requirements may have to be fulfilled before access is allowed or during access. Conditions are environmental constraints that must be considered in the process of usage decisions. Conditions are not directly associated to objects or subjects, but are based on environmental attributes. The evaluation of these conditions may take place before allowing permission to access a digital object or while the subject is using the object.

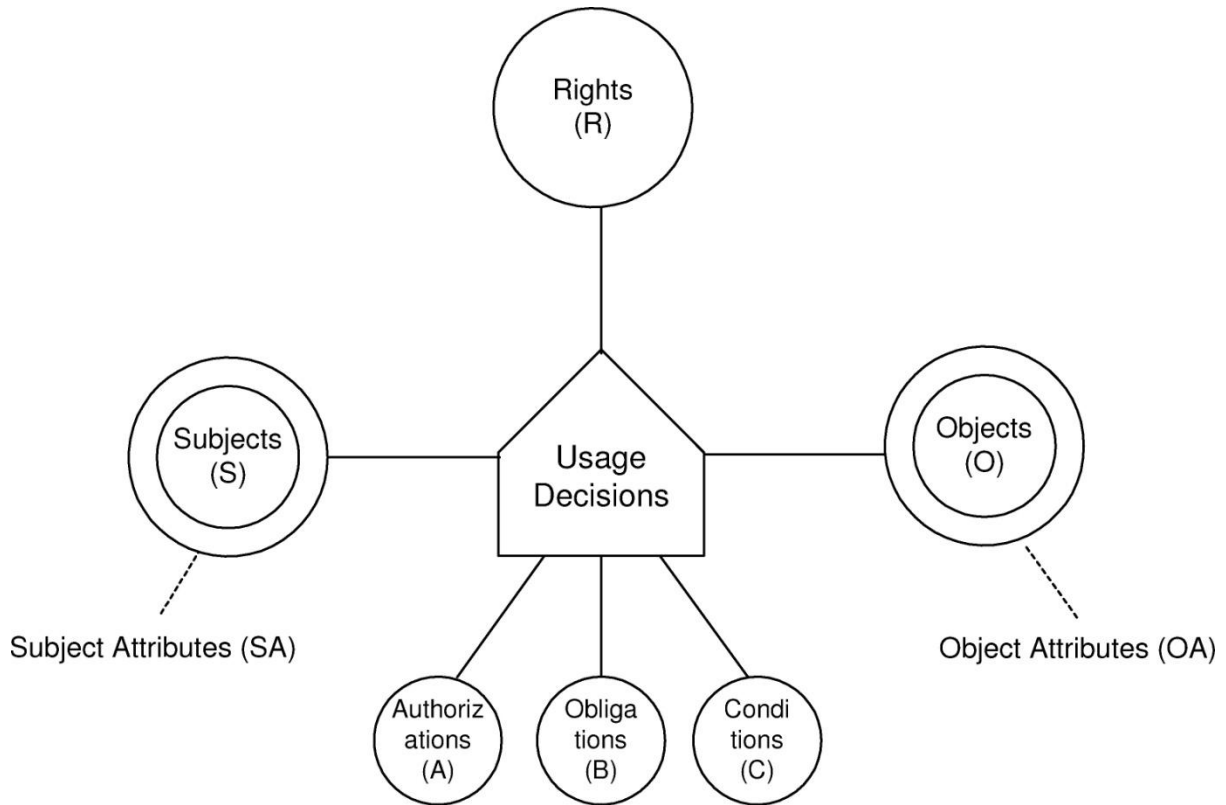


Figure 2.3 Usage Control Model Components [76]

The UCON model is conceptual and does not address specification and enforcement of policies. The UCON model has been first formalised using temporal logic of actions (TLA) [57] by Zhang et al. [100], [101]. The formal model is represented in the form of a simple state transition diagram. The main drawback of their formalism, as pointed out in [49] is that they assume a single usage process for each usage request, which adds some complexity to reason about the interactions of several concurrent usage requests. Therefore, the formalisation of UCON has been improved by Janicke et al. [49] using interval temporal logic (ITL) [71]. They focus on ongoing control. They described the UCON policies as constraints on the whole usage process rather than on a single usage process. Their formalisation assumes several (complex) usage requests as a part of a single usage process, rather than assuming only a single usage process for each atomic usage request, as in [100]. However, the concurrent enforcement of UCON policies introduces extra complexities with regards to the formal analysis of policies and the interaction between system behaviour and policy enforcement due to the atomicity of events.

CHAPTER 2. LITERATURE REVIEW

Neither TLA-based nor ITL-based UCON models deal with timing constraints involved in complex usage scenarios. This is because these underlying formalisms do not depend on metric temporal logic [54], and cannot express qualitative temporal properties. They can only support quantitative temporal operators. With qualitative temporal operators, they can specify the period of time within which events may occur, such as the interval between two consecutive events. Hence, logic-based formalisms must be based on metric temporal logic to be more suitable for formalising policies, which include timing constraints. Other approaches utilise non-logic formalisms, such as process algebra [35] as in [63] and Petri-net [79] as in [52].

The issue of concurrency has been tackled by Janicke et al. [47]. They proposed a computational model as a model of an enforcement mechanism in the form of a statechart [41], rather than as simple state transition diagrams as used in [100]. Statecharts are an extension of state transition diagrams (STDs) to support concurrency [56], and are widely used for the design of reactive systems [41]. In their model, they separate between the user, controller, and system components. Additionally, Janicke et al. [47] proposed that the issue of concurrent enforcement of UCON policies could be resolved using a static analysis of the dependencies between policy rules. Their approach is based on interleaving enforcement, where only one action can be performed at a time.

To create a veritable model and reduce the number of conflicts as the number of system states increases, it is ideal to analyse the policies using an automatic verification technique supported by tools, such as model checking and animation. Model checking is a widely used automated formal analysis technique for verifying the properties of finite-state concurrent systems. The main advantage of a model-checking tool is its ability to detect conflicts in properties in policy specifications and examine the interaction between policy rules, enforcement mechanisms, and system behaviour. On the other hand, policies can be expressed using history-based policies [1] with the ability to express the descriptions of behaviours without the need to use mutable attributes.

2.5 Insider Breach Problem and UCON

The UCON model can be seen as a comprehensive model to limit opportunities for insider data breaches prior to access being granted and after it has been granted, but it does not support the early warning and response process during ongoing insider usage. The aim of this process is effective early detection, warn against insider breaches, delay, suspend, and interrupt insider attacks in order to mitigate and counter insider breaches. This early warning and response approach must depend on insider breach scenarios, including access patterns and timing constraints.

Although UCON can be a model that is well suited with respect to the insider breach problem supporting continuous monitoring after the access has been granted, it has not completely solved the insider breach problem in a dynamic software system environment. The UCON models uses binary access control decisions, namely continue access or revoke, during ongoing usage, but the emphasis is on detecting, monitoring, and preventing insider actions. This is insufficient for dealing with insiders who pose different levels of risk. In particular, using revocation alone to interrupt insider activity is not suitable for dynamic software environments, where insider behaviour may dynamically change based on intent and context. This is a strong decision that might be a false alarm. Hence, to interrupt insider activity, there is a need to support multiple policy decisions at different risk levels and adapt dynamically to new situations.

Developing an adaptive mechanism to address insider breaches that is compatible with a dynamic software system environment is a very challenging task. The system must be able to adapt its behaviour based on insider scenarios that are continuously changing. Thus, it must respond to changes in the severity level of insider access according to different scenarios of insider breaches if it is to enforce adaptive response actions. For that reason, a security system that is utilised in a dynamic software system environment in order to enforce adaptive response actions should accordingly be policy-based. We believe that it is important to specify a well-defined policy that explicitly defines insider breach scenarios. Policies of UCON must comply with the organisation's acceptable usage policy and data protection regulations and support dynamic adaptation.

2.6 Policy Based Management

As business and security requirements and legal regulations are always changing, administrators need to review and adapt policies periodically to reflect the evolution of changes. Policy-based management [12], [90] separates between policy specifications (security requirements) and enforcement mechanisms of the system (functional requirements), which interpret and enforce these policies on the system. This separation of concern allows policies to evolve dynamically to reflect system changes without the need to re-implement the system. The adaptability of this scalable approach is essential in large-scale, dynamic environments, such as governments or health bodies.

Policy specifications, as specified in the policy language, constrain the usage of the resource, while the enforcement mechanism interprets the policy specifications to influence the system behaviour. In other words, the policy rules of the policy language are typically specified in the paradigm of ECA in order to determine the actions to be taken by the enforcement mechanism. The mechanism makes an interaction between the policy specifications and the system behaviour in order to ensure the consistency of these policies with the functional behaviour of system. Therefore, the development process of the policy-based management systems should deal with not only the policy languages, but also the ability to analyse their behaviours and check system properties [12]. Also, it is important that policy specifications and enforcement mechanisms go hand-in-hand when developing any system, and this is extremely important and difficult when developing concurrent systems with reactive and timing characteristics.

It is important to understand the types of policies that can be enforced in order to facilitate their enforcement. Security policies can be in the form of access-control policies or in the form of usage policies governed by the organisation's acceptable use policy and data protection regulations [13]. Most policy languages that have been developed are recognised within the area of access control, but there is a lack of languages governing UCON. Some concentrate on specification of privacy policies, such as P3P [26] and EPAL [9], which solve only one part of the data protection problem, where they control who can access what data for

CHAPTER 2. LITERATURE REVIEW

which purpose, but not how the data are used, once accessed. Other policy languages are based on access control, such as Ponder [29] and XACML [70] that lack formal semantics.

To increase confidence in the policy language, it is important to be based on a formal model. In security-critical systems that must protect a huge amount of privacy-critical data (e.g., e-government), it is crucial that the policy language has a sound formalism to enable the analysis of system properties, such as safety and liveness, and to detect and resolve conflicts between policies [12]. Additionally, the formal basis of a policy language plays an essential role in association with a particular enforcement mechanism model in comprehending the behaviour of the overall system to ensure compliance. The choice of formalism depends on the expressiveness of the language and the computational model of the system on which the language is based [48].

In some existing policy models, such as conventional access control, policy decisions are “static” (i.e., policy changes are triggered due to human intervention). Hence, they do not depend on the system state and changes have no dependency on time or occurrence of events. An example of such static policies is RBAC [86].

It is ideal for policies to be stateful (i.e., the policy decisions depend on the current system state and can adapt dynamically depending on changes in the system state). Some policy models require that policy decisions depend on the current and past behaviour of the system. That provides the ability to be more expressive than other policy models to describe complex policies. Examples of such stateful models are history-based access control models [1] including the policy languages Chinese Wall Policy [20] and SANTA [46], [89] as well as the mutable attributes [77] in UCON. This leads us to the notion of dynamic policies [48].

The dynamic policy can be expressed by stateful models in which policies can be adapted dynamically in response to the occurrence of events or depending on time. Hence, in the case of the occurrence of a particular event or a specified time period that has elapsed new policies will be adopted and enforced changing the existing ones. This is crucial in order to deal with insider breaches in dynamic software systems with sensitive personal data, such

as e-government, where the insider behaviour depends on time and events. Additionally, through timing-dependent policies this will allow efficient enforcement of policies at run-time.

2.7 Summary

In this chapter, we presented background regarding the nature of the insider breach problem by discussing the definitions of insider, insider threats, and the distinction between different types of insider breaches. A comparison between some existing approaches of protection from insider threats in terms of dynamic adaptation and timing constraints are made. We then investigated the usage control model (UCON), which is considered a comprehensive model that controls previous, ongoing, and subsequent resource usage. We described the differences between some existing approaches to formalising this model and their limitations. We identified the UCON model as a suitable model to address the insider breach problem. Finally, we gave an overview of the policy-based management approach by examining existing policy models, stateful and dynamic policies, and the crucial role that formal semantics of policy languages can play to enable the analysis of system properties and to detect and resolve conflicts between policies.

In the following chapter, we propose an adaptive early warning and response system architecture and computational model of this system.

Chapter 3: Adaptive Early Warning and Response System Architecture and Computational Model

Objectives

- Present the architecture of the extended UCON
- Describe the computational model for AEWRS

3.1 Introduction

In this chapter, we present a comprehensive approach that extends the traditional UCON model [76] to include interrupt policy decisions to prevent insider breaches. Interrupt policy decisions are proactive decisions at graded risk levels which can be dynamically adapted following an early warning to interrupt insider activity. The objective of this chapter is to incorporate the adaptive early warning and response processes into a usage control model. Rather than the binary access control decisions used by UCON during the ongoing usage to either continue access or revoke the user, our approach supports multiple proactive and reactive decisions. This mitigates and limits the consequences of insider breaches, or may prevent many insider activities before an actual attack takes place. The approach ensures the continuous monitoring of insider normal operations with early warning enabled, minimal human intervention, and protection from insider breaches in a timely manner.

We propose an extended UCON architecture that supports interrupt policy decisions. The main feature of the architecture is the incorporation of the interrupt policy decisions into the ongoing control to provide two layers of protection from insider breaches.

The computational model of AEWRs for a real information system is now presented. Building on policy-based management, we model the abstraction of the enforcement mechanism for the AEWRs system as regulated by the policy rules that determine the policy decision-making behaviour. Thus, the model describes a link between the policy rules and the system behaviour of AEWRs. The computational model is depicted in the form of a statechart [41], extending the model of Janicke et al. [47] that supports concurrency. The proposed model demonstrates the concurrent behaviour of a real adaptive early warning and response system. The model allows for the dynamic adaptation of policies according to the occurrence of events/actions and the passage of time.

Section 3.2 presents the extended UCON architecture and defines its components. An informal description of the computational model on which the policy specifications are based is presented in Section 3.3, along with a formal specification as a statechart in the context of policy decision-making and enforcement.

3.2 Architecture

In this section, we explain the extended UCON architecture that supports interrupt policy decisions (Figure 3.1). We define each component in the architecture, and highlight the usage decision component and the interrupt policy decision component. The two dashed ovals indicate that usage decisions occur continuously before and during usage, whereas interrupt policy decisions only happen during usage. The architecture provides a comprehensive approach which incorporates the usage decision (Section 3.2.1) to determine the access control rights with the interrupt policy decision (Section 3.2.2) to early warn, delay, suspend and respond to the insider breaches. That provides two layers of defence from insider breaches including both prevention through access control and protection through early warning, detection and response [45]. In this thesis, policies are history-based with the ability to express the descriptions of behaviours without the need to use mutable attributes.

The following subsections explain the functions of the two main components.

3.2.1 Usage Decision

This component operates as in UCON, determining whether to allow or deny access rights based on authorisation, obligation, and condition components [76]:

- **Authorisation:** Authorisation is a key functional requirement that must be fulfilled before allowing a particular right of access to a digital object. Authorisation must be evaluated both before and during access.
- **Obligation:** Obligation is used to confirm the mandatory requirements that a subject must undertake both before and during a particular usage process. These mandatory requirements may have to be fulfilled before access is allowed or during access.
- **Conditions:** Conditions are environmental constraints that must be considered in the process of usage decisions. Conditions are not directly related to objects or subjects, but are based on environmental attributes. The evaluation of these conditions may

take place before allowing permission to access a digital object, or while the subject is using the object.

- **Rights:** Rights are privileges held by a subject that can be used on an object. The subject must fulfil the required authorisations, obligations, and conditions to be allowed to access the object.

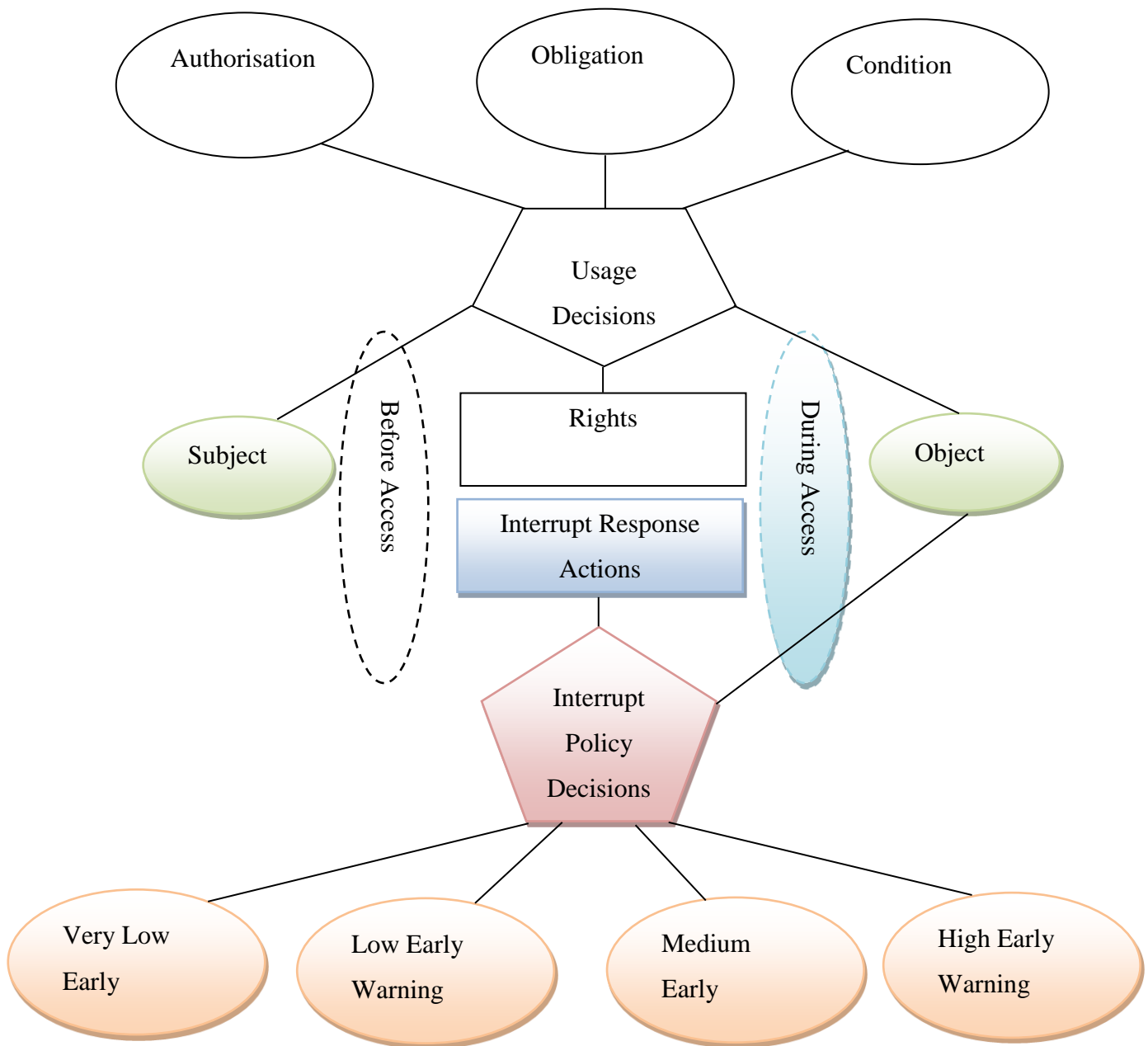


Figure 3.1 Architecture of the extended UCON

3.2.2 Interrupt Policy Decisions

Interrupt policy decisions determine what interrupt response actions will be executed depending on the level of early warning. These decisions are based on the policy decision-making and enforcement process, where policies must be enforced on the system depending on the level of early warning. The early warning level defines the system state that has been determined on the basis of prior insider behaviour. Certain guards are implemented when an early warning action is triggered. These are known as interrupt policy decisions, because they are used to interrupt the insider activity during ongoing usage when an early warning level is operational. Our approach allows for the dynamic adaptation of policies according to both the occurrence of events/actions and the passage of time.

3.2.2.1 Early warning levels

The early warning levels are determined by the early warning level actions in the policy rules. The policy rules are specified in the *Event-Condition-Actions (ECA)* paradigm in order to determine which actions are to trigger the enforcement mechanism. The insider breach scenario detected by AEWRs triggers the corresponding early warning level action specified in the policy rules. When triggered, that action causes the early warning guard to become “true”. The early warning level action corresponding to a policy rule distinguishes the different early warning risk levels. These actions determine different risk levels: high, medium, low, or very low.

To signal a particular early warning level, we use the access history matching past insider behaviour (Chapter 4). The access history is expressed in the privacy breach scenario of the early warning policy rules using our PBSL specification.

1. Very Low Early Warning Level

This level is determined by the very low early warning policy action of the corresponding policy rule. This very low early warning rule is triggered by a breach scenario at a very low level. This implies there is a small chance of insider breach, and that the threat of this potential suspicious activity would not affect the

information asset, as the insider behaviour is of very low risk. The system should be aware of this very low risk activity, as the insider behaviour may change to higher-risk activities in the future.

2. Low Early Warning Level

This level is determined by the low early warning action of the corresponding policy rule. This low early warning policy rule is triggered by a low-level breach scenario, which implies that there is a chance of insider breaches that could potentially affect sensitive personal information assets. This suspicious activity requires a proactive response action. Therefore, the system investigates this activity, and conducts real-time monitoring of ongoing usage. This allows for a reactive response if the risk level increases to medium or high.

3. Medium Early Warning Level

This level is determined by the medium early warning action of the corresponding policy rule. The medium early warning policy rule is triggered by a breach scenario at a medium level, implying an increased likelihood of a malicious insider breach that could potentially damage or disclose sensitive personal information assets. The system must respond proactively to this potential malicious activity, protecting the asset from the most serious impacts of an insider breach.

4. High Early Warning Level

This level is determined by the high early warning action of the corresponding policy rule. The high early warning policy rule is triggered by a high-level breach scenario, signifying that a direct and malicious insider breach has been detected. The system must immediately take the most strict response actions to protect sensitive personal information assets from the critical impact of an insider breach.

3.2.2.2 Interrupt Response Actions

An interrupt response action must be performed when an early warning level has been triggered, with the aim of responding to insider breaches in a timely manner. In other words, the interrupt response action is governed by an early warning level guard that must be conditionally executed when the guard is true.

There are four categories of response actions, one for each level of early warning. These are described as follows:

1. Response action based on Very Low Early Warning

If the AEWRS detects an insider scenario at a very low early warning level, then the very low early warning guard would hold. This triggers the skip response action, in which the request is executed after one time unit, and the system skips to the user's normal mode of operation. There is no clear indication that the insider was misusing the information asset; thus, this response could also be used for false positives (false alarms). Although the request is executed, the action is logged.

2. Response action based on Low Early Warning

If the AEWRS detects an insider scenario at a low early warning level, then the low early warning guard is evaluated to true. Consequently, a response action is triggered that delays access for a period of time. This slows the insider's progress by increasing the insider's activity time. In addition, this increases the probability of early detection of insider breaches (with fewer false alarms) if subsequent investigation is carried out during the delay period of time. Further, the user can initiate other requests as part of the current usage process.

Delay action can be more affective when allowing timely intervention by the system administrator following the early warning to investigate the incident within the delay period and respond in a timely manner to prevent the breach. If the system

administrator has been notified of insider activity, and detects that this is a direct breach within some duration t , then he/she can make an early decision and take stricter response actions within the delay duration. Otherwise, in the case of a timeout, the system automatically reverts to the user's normal operation.

In the case of further interrupts at higher early warning levels, namely medium or high under the continuous monitoring, the system takes a stricter response action, such as suspending operations while waiting for an answer from the user or automatically aborting operations by interrupting the delay period.

3. Response action based on Medium Early Warning

At the medium level, the system responds proactively by waiting for a user event to protect information assets from the potential high impact of an insider breach. If the AEWRS detects an insider privacy breach at the medium early warning level, the suspend response action is triggered, causing access to be suspended until some event occurs, e.g. an answer from the user, or the system times out. The main feature of this response is to suspend the insider activity, and impede the insider's progress to get unauthorised access. The suspend action provides a stronger response than the delay action in a manner that it complicates the insiders progress by requiring him/her to use stronger mechanisms in order to gain access to the information assets within that suspension period. Within the suspension period, the user can neither initiate other requests within the current usage process nor initiate a new process.

Using stronger mechanisms to complicate the insider's progress within the suspension period can be developed. For example, the system challenges the user by asking for re-authentication to check whether they have legitimate authorisation to access the information assets they are trying to access. The system waits for t units of time for some event, such as an authentication decision, to occur. As soon as the event occurs, e.g. re-authentication is verified, the suspension period will be interrupted and the user's normal operation resumed. If t time units elapse and no event, e.g. re-authentication decision, has occurred or the event returns a "false" evaluation, e.g. re-

authentication is not verified, and then the suspension period will be interrupted and automatically changed to an aborted decision to terminate the access.

A suspend action can increase the likelihood of early detection of insider breaches if further investigation is carried out by the system administrator following the early warning within suspension period. The system can raise an alert to the system administrator requesting them to investigate the insider activity. If the system administrator detects that this activity is a direct breach within the suspension duration t and before any event occurs, then the administrator can take stricter response actions, such as aborting the operation, according to the level of severity and confidence of the insider breach. In AEWRS system, we do not describe such implementation details of any interactions between the system and the user throughout this thesis, as we leave them as an implementation decision.

4. Response action based on High Early Warning

This action disconnects the insider from the system, and immediately aborts the access process. If AEWRS determines that an insider privacy breach scenario is at the high early warning level, an abortive response action will be generated to interrupt the user's normal operation. At this time, the user can neither initiate other requests within the current usage process, nor initiate a new usage process.

Complex Response Actions

Further complex response actions can be triggered within the delay or the suspension period. These allow two or more response actions to run in parallel or in sequence within the delay or suspend period. For example, once the very low early warning level is determined following some privacy breach scenario, the access process will be delayed while, in parallel, video surveillance monitoring is performed. In addition, an alert response action to the system administrator could be triggered after this delay, namely in sequence, to achieve further investigation and ensure awareness of the potential insider activity. Furthermore, triggering one response action may lead to

other response actions, such as delay leading to an alert. Examples of complex response actions within the delay period include an Alert and Video Surveillance, while examples of complex response actions within the suspension period include challenge questions and the use of stronger mechanisms such as multi-factor authentication.

Complex response actions can be triggered based on the high early warning level. For example, once the very low early warning level is in operation, such as when bulk data is accessed, the skip action will be triggered. This does not do anything at that time. If the user has transferred the data by FTP or USB drive, which determines the high early warning level, then the user's normal operation process will be interrupted and the data transmission will be blocked. Next (i.e. in sequence), the user's session will be disconnected. Examples of complex response actions based on high early warning level are Disconnect, Block, and Block Data Transmission. However, in AEWRS, we do not include the details of complex response actions as they are out of scope for this thesis.

3.3 Computational Model

In this section, we present the computational model for AEWRS within a real information system. The abstract components of the system, their behaviour, and interactions are depicted. Building on policy-based management, we represent the abstraction of the enforcement mechanism model for the AEWRS system as regulated by the policy rules that determine the policy decision-making behaviour. Policy-based management separates the policy specifications and the enforcement mechanisms of the system. Policy specifications constrain the usage of the resource, while the enforcement mechanism interprets the policy specifications to influence the system behaviour. The policy rules in PBSL (Chapter 4) are specified in the ECA paradigm to determine which actions should trigger the enforcement mechanism. The computational model applies to the specification of the policies and policy enforcement to express multiple policy decisions at different risk levels during run-time and dynamically adapt policies to new situations. In particular, the model allows for the dynamic adaptation of policies according to the occurrence of events/actions and the passage of time.

The behaviour of the adaptive early warning and response controller (AEWRC) and its interaction with other system components are detailed in Figure 3.2 as a statechart. Statecharts are an extension of state transition diagrams (STDs) to support concurrency [41], and are widely used for the design of reactive systems [42]. The dashed line represents the concurrency of the system components. The nodes are referred to as states and the edges are referred to as transitions. The labels on the transitions are based on the form *Trigger [Guard] /Action*, where transition has taken once the trigger event occurs; if the guard condition evaluates to true, then the action is executed. If the insider breach scenario is detected by AEWRC, then the corresponding early warning level is triggered according to the policy rules. The outcome of the policy rules, namely early warning level, will affect the behaviour of the computational model by enabling the corresponding state transitions, namely interrupt response actions, changing the ongoing control and system states.

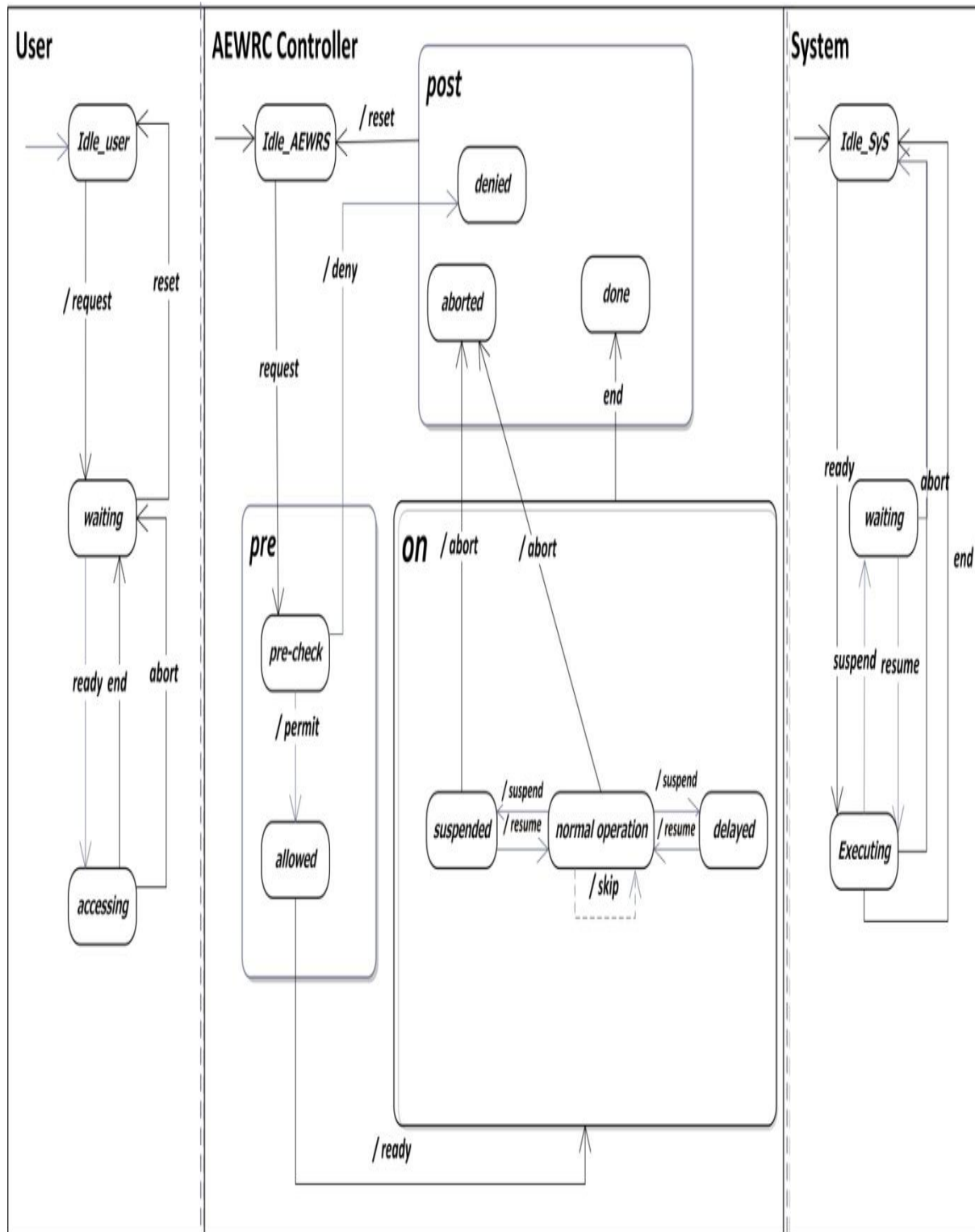


Figure 3.2 Computational model

The AEWRc policies in Figure 3.2 are stateful, i.e. they adapt dynamically depending on changes in the system state. The system state can be changed on the basis of the system triggers caused by a user accessing an object. As represented diagrammatically in Figure 3.3, the AEWRc can be in one of four states: Normal Operation, Delayed, Suspended, or Aborted. These states can be reached by interrupt policy decisions. Interrupt policy decisions determine what response actions to execute depending on the level of early warning and/or due to timeout, causing the controller state to change. Response actions depend on the effect of early warning policy rules, acting as a policy enforcement mechanism. The AEWRc enforcement mechanisms enforce policies on the system. The decisions of the policy rules depend on the insider breach scenario (access patterns and timing constraints) considered in the policy decision-making.

To specify, animate, and verify the interrupt policy decisions, we represent the computational model for the AEWRc at two levels of abstraction. In Figure 3.3, the computational model for AEWRc is depicted at a lower level of abstraction. This displays an abstract representation of interrupt policy states which can be dynamically changed by certain response actions. The response actions cause a transition from the current state (precondition state) to the next state (postcondition state) when the early warning level guard is “true”. We support guarded enforcement to ensure mutual exclusion, where no conflicts can occur between two policies whose rules fire at the same time. The policy rules express the corresponding early warning actions for some insider privacy breaches through the PBSL (Chapter 4). Through our formal framework, the policy rules and enforcement mechanisms must be complete and free from conflicts with regard to system properties. The ability to verify these properties and ensure consistency between these high-level goals with low-level policy rules and enforcement mechanism behaviour is described in Chapter 7.

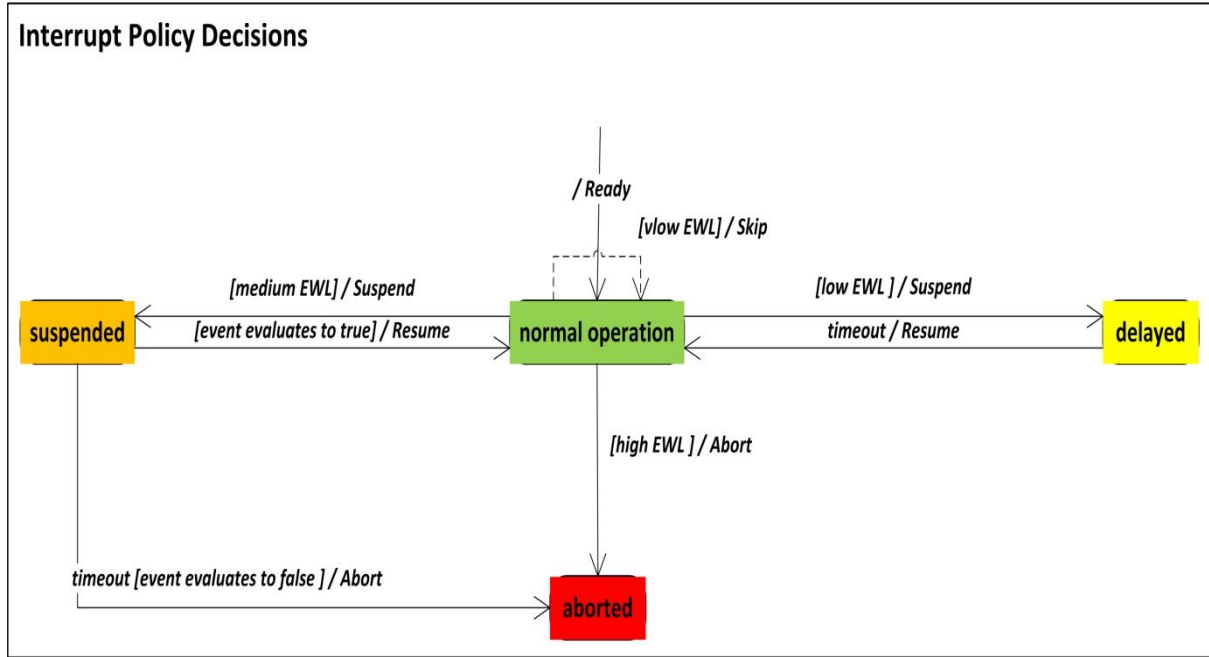


Figure 3.3 AEWRC

An informal description of the three abstract components of the computational model in Figure 3.2 and a detailed description of AEWRC (Figure 3.3) are as follows:

3.3.1 User model

A user process is represented by a user. This process has three states: *idle_user*, *waiting*, and *accessing*. The initial state *idle_user* corresponds to the state in which the user process started. By raising the *request* event, the user process will enter the *waiting* state, where the request remains until either the access request is processed and *reset* or *ready* to execute. The *accessing* state lasts from when the user process starts accessing object until access is completed with the *end* event or access is interrupted with the *abort* event.

3.3.2 AEWRC model

The controller in Figure 3.2 is initially assumed to be in the *idle_AEWRC* state. Upon a *request* event being received, the controller enters the *pre-check* state, which represents the standard access control policy, in order to determine binary access control decisions. These decisions either allow or deny access, depending on the policy. When access is permitted, the

controller generates a *permit* event and enters the *allowed* state. The *ready* event is then generated to indicate that the actual access can take place. Otherwise, if access is denied, the *deny* event is generated, and the controller moves into the *denied* state. The *accessing*, *on*, and *executing* states then synchronise on *ready* event.

Figure 3.3 represents the AEWRC model, where the response actions are guarded by the early warning levels. In the *normal operation* state, and when the guard medium early warning level—*medium EWL*—holds, the controller can suspend access, moving into the *suspended* state and wait, which causes the access to cease until an event or a *timeout* occurs. When an event occurs, access is resumed by raising a *resume* event and returning to the *normal operation* state. Otherwise, an *abort* event is generated if no suitable event occurs within the suspension period or the system times out. On the other hand, when the guard low early warning level—*low EWL*—is true, the controller will raise a *suspend* event and move into the *delayed* state. The *delayed* state is a sleeping state that delays access for some period of time, and then resumes operation automatically by generating a *resume* event and moving into the *normal operation* state. In the *normal operation* state, the controller can raise an *abort* event that terminates the access when the guard high early warning level—*high EWL*—holds. When the very low early warning level—*very low EWL*—is evaluated as true, the *skip* event is generated. It is worth noting that this event, depicted as a dotted line, is just a null event.

3.3.3 System model

Access to objects is handled in the system process. It is assumed that the system is initially in the *idle_SyS* state. Once the *ready* event indicates that the access request can be executed, the system enters the *executing* state by generating an *execute* event. This synchronises the *accessing* and *executing* states during the access, although the specific behaviour of the user process and the system in these states is not explicitly defined. Upon a *suspend* event, the system process will move from the *executing* state into the *waiting* state until either a *resume* event moves it back to the *executing* state or an *abort* event moves the system to the *idle_SyS* state. The effects of *suspend* and *resume* events in the system are transparent to the user process, so they trigger changes in the system states and policy specifications.

The computational model demonstrates the abstract concurrent behaviour of an AEWS for a real information system, where concurrent data access requests can be handled prior to usage using access control decisions, and then once usage is allowed by both ongoing control and the system execution.

3.4 Summary

In this chapter, we presented an extended UCON architecture that supports interrupt policy decisions (Section 3.2). These multiple proactive policy decisions operate at different risk levels, and can be dynamically adapted to interrupt insider activity upon an early warning being triggered. This architecture provides a comprehensive approach which incorporates usage control process to determine the access control rights with the adaptive early warning and response process to early warn, delay, suspend, and respond to the insider breaches. The main feature of the architecture is the incorporation of the interrupt policy decisions into the ongoing control to provide two layers of protection from the insider breaches.

We then proposed an AEWS computational model for a real information system in Section 3.3. The computational model was depicted in the form of a statechart. This demonstrated the concurrent behaviour of an AEWS, where concurrent data access requests can be handled. Building on policy-based management, we represented the abstraction of the enforcement mechanism model for an AEWS system regulated by the policy rules of PBSL to determine the policy decision-making behaviour. Thus, it describes a link between the policy rules and the system behaviour of AEWS. The model showed how policies can be adapted dynamically over, both, the occurrence of events/actions and the passage of time.

The following chapter describes PBSL, which is based on the computational model, and Chapter 5 presents the PBSL formal operational semantics. Chapter 6 formalises the enforcement mechanisms model, and Chapter 7 shows that these policy and enforcement mechanism specifications can be verified against the system properties.

Chapter 4: PBSL – Linguistic Support

Objectives

- Describe PBSL
- Demonstrate the use of PBSL by examples

4.1 Introduction

This chapter defines a privacy breach specification language (PBSL) based on the computational model defined in Chapter 3. The objective of this chapter is to specify privacy breach scenarios and the corresponding early warning levels in the form of policy rules. Rule-based languages are common in event-driven systems, and particularly in active database systems such as SNOOP [24], SAMOS [36], and ODE [37], which typically follow the ECA paradigm. Under ECA, once an event occurs, if the condition evaluates to true, the action is executed. Based on ECA rules, our PBSL determines the actions to trigger the enforcement mechanism. The standard ECA language [78] has been extended with some constructs, event-supporting patterns (e.g. sequence, parallel), and timing constraints, as discussed in detail throughout this chapter.

Policy specification is the process of expressing what response actions are triggered when some detected privacy breach scenario triggers a certain early warning level. Policy rules form the basis of PBSL specifications. The specification of policy rules is one of the main challenges of capturing privacy breach behaviour, and allows for early warnings at run-time. Privacy breach scenario refers to a combination of events with some associated contextual information which identify the potential insider privacy breach, while the corresponding early warning relates to the decisions resulting from the breach scenario.

Ideally, a policy specification language should meet the following criteria:

- **Expressiveness:** The language is able to express a wide range of complex events and timing constraints in a comprehensive scenario, and can clearly specify a wide range of complex policies from various application domains.
- **Formal semantics:** The language is based on sound formal semantics. This formal underpinning ensures the system description behaves unambiguously, and allows for the analysis of the policy specifications and their enforcements and the verification of system properties such as safety and progress.

- **Simplicity:** The constructs of the language should have a simple meaning that have foundations in existing work. This makes the language readable (easy to understand) and writable (easy to specify).
- **Practicality:** The language can be implemented using an efficient mechanism in a practical setting.

This chapter gives a detailed description of PBSL. The syntax and informal descriptions are presented in Section 4.2. The chapter ends with some examples of common insider privacy breach scenarios in Section 4.3.

4.2 Privacy Breach Specification Language

In this section, we describe the PBSL constructs based on the computational model defined in Chapter 3. Policy rules are an integral part of the system specification. They can capture the access history and timing constraints in the privacy breach scenario using a high-level description of the system behaviour that triggers the corresponding early warning level. The objective of PBSL is to specify privacy breach scenarios and the corresponding early warning levels.

The PBSL proposed in this thesis is based on the well-known ECA language. In PBSL, an event is used to represent the detection of an insider privacy breach, while conditions are used to constrain the context associated with the privacy breach behaviour. Actions are used to model the corresponding early warning level. The early warning level is then used by the enforcement mechanism to enforce the policy in AEWS.

PBSL expresses privacy breach scenarios and the early warning level corresponding to the insider privacy breach and various timing constraints. To specify privacy breach scenarios composed of temporally related events, an event language is proposed that can

express complex events. To specify the relationship between events, sequence, disjunction, parallel, negation, and implication operators are used. The language can also specify repetition. To constrain the period of time in which events may occur, various timing constraints are proposed to specify the interval between two events and the duration of the privacy breach scenario.

4.2.1 Policy Rules

A policy rule expresses the response action for some insider privacy breach scenario. The rule consists of a privacy breach scenario and a corresponding early warning level. The privacy breach scenario describes a combination of events with some associated contextual information that, when detected by the AEWRC, trigger the corresponding early warning level. The action of a policy rule defines the early warning decision that triggers the enforcement mechanism of AEWRC. The corresponding early warning level of a rule distinguishes between different severity levels of the insider access. These early warnings can be high, medium, low, or very low level.

Our policy rule takes the general form:

PolicyRule \triangleq {*Privacy breach scenario*} \mapsto *Signal Early Warning (L)*

PolicyRule \triangleq

$$\left\{ \begin{array}{l} \text{Event: event pattern} \\ \text{Condition : attributes constraints} \wedge \\ \text{timing constraints} \end{array} \right\} \mapsto \text{Action: Signal Early Warning (L)}$$

Where

- **PolicyRule** is the keyword of the policy rule name which introduces the policy rule, followed by the privacy breach scenario in parentheses. The privacy breach scenario

is separated from the early warning level by an arrow, \mapsto . This arrow denotes the implication operator, leading to the corresponding early warning level, when a privacy breach scenario is detected.

- ***event pattern*** is the combination of events that match the event history, under the keyword **Event**.
- ***attributes constraints*** express the context associated with the event pattern in the privacy breach scenario, under the keyword **Condition**.
- ***timing constraints*** is the temporal expression constraining when the event pattern happened, under the keyword **Condition**.
- ***Signal Early Warning (L)*** is the early warning level triggered in response to the privacy breach scenario, under the keyword **Action**. The early warning level **L** can be (*high*), (*medium*), (*low*), or (*vlow*).

The policy rules describe which event patterns, attributes, and timing constraints must be true to trigger the action that signals an early warning level. This action causes the early warning guard EWL to become true in the computational model (Section 3.3), triggering the enforcement mechanism. Consequently, the enforcement mechanism influences the actual system state by triggering interrupt response actions which are guarded by the early warning level.

PBSL rules express some high-level constructs, and neglect unnecessary details at the implementation level. Policy rules can express the current and past behaviour of the insider in the system, allowing complex events that match the event history to be expressed. The following subsections explain the syntax and informal descriptions of event patterns observed by the AEWRC.

4.2.1.1 Event patterns

The notion of an event represents a change in the system. Events can be primitive or complex. Primitive events are instantaneous, i.e. they occur at a single point of time, within a context, and atomic, i.e. they cannot further broken up and occur completely or not. Complex events (also known as compound events) occur when multiple primitive events occur according to some event pattern. The complex events occur within a time interval rather than at a single point of time.

The event pattern is characterised by the event type and order. Event types are captured in the privacy scenario of the policy rules, or as external events and timing events. The events captured in the privacy scenario refer to those that can be controlled by the system, and use the event description and context that trigger the policy rule. External events are observable by AEWRC from external systems. Further, a timing event is an explicit clock tick.

An Event is characterised by its name associated with a set of attributes. The event's attributes can be related to the information about the event itself, such the object and command name; or related to the contextual information relevant to the event, such as the session number. These attributes are constrained by the condition components of PBSL policy rule. To capture the event's attributes, variables are bound to the structural and contextual information about the event, which can be utilised in the condition component of PBSL 4.2.1.2.

A privacy breach scenario is expressed by a combination of events and the contexts that trigger the policy rule. Table 4.1 defines the syntax of the event pattern, and then the notation is described, where:

- $H = \{e_1, e_2, \dots, e_m\}$: history of events.
- Each event e_i may carry attributes.
- $e_i.attr_k$ denotes the value of the attribute $attr_k$ of an event e_i .
- $e_i^n : e_i \times e_i.attr_k \rightarrow \{\text{true}, \text{false}\}$, where n is the number of occurrences.

- $t, n \geq 1$.

Table 4.1 Syntax of *PBSL*: event Patterns

Construct	Description
e_1^n	Repetition (number of occurrences)
$e_1 @ t$	Happens at a particular point of time
$e_1 ; e_2$	Sequence of events
$e_1 e_2$	Events in parallel
$e_1 \vee e_2$	Alternative events
$\neg e_2$	Absence of event
$e_1 \rightarrow e_2$	Implication of events

Basic event constructs

An informal description of the basic event constructs is as follows:

- $e_1 @ t$. $e_1.time = t$

This represents an event e_1 happens at a particular point of time t .

- $e_1 ; e_2$

This scenario is the combination of consecutive events. The instances of events must be specified in the scenario in the order they occur. $e_1 ; e_2$ starts with event e_1 and is followed by event e_2 .

For example, *queryBulkData(D); transferBulkData(D)*.

- $e_1 \parallel e_2$

This is a parallel scenario. It represents a combination of events which happen at the same time, i.e. the events have identical occurrence times. $e_1 \parallel e_2$ means that events e_1 and e_2 happen simultaneously.

- $e_1 \vee e_2 \triangleq \text{if } e_1 \parallel e_2 = \text{true then } e_1 \vee e_2 = \text{false}$

This describes alternative events. This scenario is detected when e_1 or e_2 happens.

- $\neg e_3 \triangleq \neg e_3 \notin H$

That refers to the absence or non-occurrence of an event during the interval in which a sequence of two or more events occurs. For example, $(e_1; e_2) \wedge \neg e_3$ denotes that event e_3 does not happen during the period when events e_1 and e_2 happen. This implies that, if e_3 happens in the context of a complex event (namely $(e_1; e_2)$), the privacy breach scenario is incomplete. The absence operator \neg is usually used with any order operator \wedge (a derived event construct).

For example, ***Access***₁(*user*, *01*); ***Access***₂(*user*, *02*) \wedge ***Delete***(*user*, *01*).

Derived Event Constructs

Some derived event constructs are as follows:

- $e_1^n \triangleq e_1^{n-1}; e_1$

This denotes that event e_1 must happen n times for the scenario to be detected.

For example, ***Access***₁(*user*, *01*)¹⁰.

- $e_1^{\geq n} \triangleq \exists k (k \geq 1 \wedge e_1^k \wedge n \leq k)$

This denotes that event e_1 must happen a minimum of n times for the scenario to be detected.

- $e_1^{\leq m} \triangleq \exists k (k \geq 1 \wedge e_1^k \wedge k \leq m)$

This denotes that event e_1 must happen a maximum of m times for the scenario to be detected.

- $e_1^{n,m} \triangleq \exists k (k \geq 1 \wedge e_1^k \wedge n \leq k \leq m)$

This denotes that event e_1 must happen between a minimum of n times and a maximum of m times for the scenario to be correct and complete.

- $e_1 \wedge e_2 \triangleq (e_1; e_2) \vee (e_2; e_1) \vee e_1 || e_2$

This scenario is composed of two or more events that can happen in any order (i.e. the order is not essential). The scenario is detected when both e_1 and e_2 happen. However, e_1 and e_2 may happen concurrently.

- $e_1 \rightarrow e_2 \triangleq \neg e_1 \vee e_2$

This denotes that event e_1 implies that another event e_2 happens.

When the events are detected, certain conditions need to be evaluated, as described in the following subsection.

4.2.1.2 Attributes Constraints

Attributes constraints are used to constrain the events and their contexts that should occur in the privacy breach scenario of the policy rule. Attributes constraints are Boolean expressions. They can be defined using comparison operators either on the attributes of one event or on attributes of multiple events belonging to the same scenario, or can be defined using

predicates. The constraints that need to be evaluated are restricted to the policy rules of those events that have occurred.

Every event has domains of event attributes that define which variables apply to each event. The domain represents the sets of variables. The event pattern often applies to the same domain of event attributes. The following table explains the syntax and informal descriptions of attributes constraints.

- $v1, \dots, vn \in Var$: set of variables.

Table 4.2 Syntax of *PBSL*: Attributes Constraints

Construct	Description
$v \sim x$	$\sim \in \{<, \leq, >, \geq, =, \diamond, \text{in}\}, v \in Var, x \text{ is a constant}$
$v1 \sim v2$	$\sim \in \{=, \diamond\}, v1 \in Var \text{ and } v2 \in Var$
$P(v1 \dots vn)$	Predicate for history of events from different domains
$b_1 \wedge b_2$	Complex constraints, b_1, b_2 are two Boolean expressions

- $v \sim x \triangleq e_i . attr_k \sim x$

This denotes the relationship between a variable or an attribute and a constant for the current event from the same domain of event attribute names. The attributes of an event are prefixed with the event type and a dot.

- $v1 \sim v2 \triangleq e_m . attr_k = e_n . attr_k$

This denotes the relationship between variables or attributes for historic events including two or more from the same domain of event attribute names. This compares

an event in the current state with previously observed events. The attribute of the event is prefixed with the event name and a dot.

- $P(v1 \dots vn) \triangleq \neg (\neg P(v1 \dots vn))$

This predicate denotes the relationships between variables or attributes for historic events from different domains of event attribute names. This predicate might be used as an alternative way to relate attributes or variables in the same domain. The attribute of the event can be prefixed with the event name and a dot.

$$P(e_n.attr_k, e_n.attr_l, e_n.attr_m)$$

- $b_1 \wedge b_2 \triangleq \neg (\neg b_1) \wedge \neg (\neg b_2)$

This denotes complex attributes, predicate, or timing constraints expressed in combination as Boolean expressions.

4.2.1.3 Timing Constraints

To constrain the period of time within which events may occur, timing constraints have been introduced in PBSL. Under these constraints, we can specify the breach scenarios that may occur during the ongoing usages. These constraints imply that events must occur within a number of clock time units, as a time interval (i.e. time period between two events) or the duration of a scenario. The following defines the syntactic elements of the timing constraints together with their informal descriptions.

Table 4.3 Syntax of *PBSL*: Timing Constraints

Construct	Description
<i>Duration</i> (d)	Time period within which the event patterns can happen
<i>Interval</i> (t_{min}, t_{max})	Minimum and maximum interval between two events
<i>Timeout</i> (d)	Wait duration within which an action of a policy rule must be triggered
<i>Before</i> (e_1, t)	Event happens earlier than a particular time
<i>Every</i> (e_1, t, e_2)	Multiple times periodically

- ***Duration*** (d)

This represents the time period d within which the event pattern can happen from the start of a scenario.

For example, $\left\{ \begin{array}{l} \text{Event: } e_1^n \\ \text{Condition: } \textit{duration}(180) \end{array} \right\}$ states that e_1 must happen n times

within the duration of 180 units for the scenario to be detected.

- ***Interval***(t_{min}, t_{max})

This denotes the minimum t_{min} and maximum t_{max} time period between two events, per scenario.

For example, $e_1; \textit{interval}(60, 180); e_2$ states that e_1 is followed by e_2 when the time interval is between 60 and 180 time units.

- ***Timeout* (d)**

This represents the time period d within which an action of a policy rule must be triggered.

For example, $\left\{ \begin{array}{l} \text{Event: } e_1^n \\ \text{Condition : } \textit{timeout} (3) \end{array} \right\}.$

- ***Before* (e_1, t)**

This represents a period that starts relatively earlier than a particular time t at which e_1 happens.

For example, ***Before* ($\textit{Access}_1, 180$)** states that ***Access*₁** event happens after the time period of 180 time units has passed.

- ***Every* (e_1, t, e_2)**

This describes an interval that happens periodically (i.e. every t time units) between events e_1 and e_2 .

For example, ***Every* ($\textit{Access}_1, 30, \textit{Access}_2$)** states that it happens every 30 time units (periodic time interval) after ***Access*₁** happens until ***Access*₂** happens.

The formal semantics of the timing constraints are described in the next chapter.

4.2.1.4 Early Warning Actions

Early warning actions represent operations that must be triggered once a privacy breach scenario has been detected. At the syntax level, the action is to signal an early warning level. The outcome of this action is taken by the enforcement mechanism of AEWRC.

Table 4.4 Syntax of *PBSL*: Early Warning Actions

Construct	Description
<i>Signal Early Warning (L)</i>	$L \in \text{high, medium, low, vlow}$

- ***Signal Early Warning (L)***

This action signals an early warning level when a privacy breach scenario is satisfied. The early warning action is followed by an argument **L** to denote the High, Medium, Low, or Very Low level, using the keywords (**high**), (**medium**), (**low**), and (**vlow**), respectively.

4.3 Some examples of PBSL usage

This section provides illustrative examples of PBSL specifications with regard to some real-world situations and examples of insider privacy breaches. These examples have been chosen based on the categories of insider privacy breach scenarios in Section 2.2.3.4.1 (Chapter 2). These include data theft, masquerading, data inference, access for unauthorised purposes, and access without consent. We informally describe the syntactic elements of PBSL using such examples. Some of these will be utilised in our case study evaluation in a subsequent chapter.

4.3.1 Example 1 (Data theft)

Insider Breach Scenario: An organisation is vulnerable to an ‘operator’ who wishes to commit an identity theft. The operator could submit a query to get the ‘phone’, ‘age’, and ‘income’ details of ‘tax payers’ with an ‘age’ of, for example, 65 and an ‘income’ of more than £1 million [97]. The insider privacy breach happens only during normal working hours (between 09:00 and 17:00), as the operator is not allowed to work outside these hours. The

breach happens when the number of records accessed exceeds 100, and the duration of this scenario is no more than 180 time units.

PolicyRule_DataTheft_1 \triangleq

$$\left\{ \begin{array}{l} \text{Event: } \text{QueryBulkData}(\text{user}, \text{object}, \text{command}, \text{session}) \\ \\ \text{Condition: } \text{QueryBulkData.user} = "U1" \wedge \\ \text{QueryBulkData.object in "phone, age, income"} \\ \text{QueryBulkData.commnad} = "read" \wedge \\ \text{count}(\text{QueryBulkData.object}) > 100 \wedge \\ \text{allowed}(\text{QueryBulkData.user}, \text{QueryBulkData.object}, \text{QueryBulkData.action}) \wedge \\ \text{Between}(9, 17) \wedge \\ \text{duration}(180) \end{array} \right\}$$

\mapsto **action:** *Signal Early Warning (vlow)*

In the above policy rule, the privacy breach scenario is represented by the event *QueryBulkData* and a set of conditions and timing constraints, leading to the determination of the very low early warning level.

A further possible privacy breach scenario could occur when the *TransferData* event (i.e. user transfers data by FTP) is executed after the *QueryBulkData* event by the same user on the same object within the same session. This would lead to a high early warning signal according to the following policy rule:

PolicyRule_DataTheft_2 \triangleq

$$\left\{ \begin{array}{l} \text{Event: } \text{QueryBulkData}(\text{user}, \text{object}, \text{command}, \text{session}) \\ \quad ; \text{TransferData}(\text{user}, \text{object}, \text{command}, \text{session}) \\ \\ \text{Condition: } \text{QueryBulkData.user} = \text{TransferData.user} \wedge \text{QueryBulkData.object in "phone, age, income"} \\ \quad \wedge \text{QueryBulkData.command} = "read" \wedge \text{count}(\text{QueryBulkData.object}) > 100 \wedge \\ \quad \text{TransferData.object in "phone, age, income"} \wedge \\ \quad \text{TransferData.command} = "transfer by FTP" \wedge \\ \quad \text{QueryBulkData.session} = \text{TransferData.session} \wedge \\ \text{allowed}(\text{QueryBulkData.user}, \text{QueryBulkData.object}, \text{QueryBulkData.command}) \wedge \\ \quad \text{between}(9, 17) \wedge \text{duration}(200) \end{array} \right\}$$

\mapsto **action:** *Signal Early Warning (high)*

4.3.2 Example 2 (Masquerading)

Insider Breach Scenario: An organisation's acceptable usage policy states that users are not allowed to access an object using two simultaneous connections. Consider a user who logs in to the system to perform routine duties by accessing an object. A breach scenario occurs when another person in another place uses the same user account and password to login to the system and access the same object the actual user is authorised to access. The breach happens when two people use the same login credentials from different machines and access the same object at the same time outside of working hours (17:00–08:00).

PolicyRule_Masquerading_1 \triangleq

$$\left\{ \begin{array}{l} \text{Event: } Access_1(user, object, command, Session, IP) \parallel \\ \quad Access_2(User, object, command, Session, IP)) \\ \text{Condition: } Access_1.user = Access_2.user \wedge Access_1.object = Access_2.object \wedge \\ \quad Access_1.session \neq Access_2.session \wedge Access_1.IP \neq Access_2.IP \\ \quad \wedge Access_1.command = Access_2.command \wedge \\ \quad \quad \wedge Between(17, 8) \\ \wedge allowed(Access_1.user, Access_1.object, Access_1.command) \end{array} \right\}$$

\mapsto **action:** *Signal Early Warning (low)*

In the above policy rule, the privacy breach scenario represents two events $Access_1$ and $Access_2$ that can be distinguished by comparing their attributes, such as the session, object, and user. For instance, $Access_1.IP \neq Access_2.IP$ means that the IPs of the two events are not the same, $Access_1.user = Access_2.user$ refers to using the same user credentials, and $Access_1.object = Access_2.object$ denotes that the same object is being accessed.

A further insider scenario might include the object being updated by one of the users as follows:

PolicyRule_Masquerading_2 \triangleq

$$\left\{ \begin{array}{l} \text{Event: } (Access_1(user, object, command, session, IP) \wedge \\ Access_2(user, object, action, Session, IP)) ; update(user, object, command) \\ \text{Condition: } Access_1.user = Access_2.user \wedge Access_1.object = Access_2.object \wedge \\ Access_1.session \neq Access_2.session \wedge Access_1.IP \neq Access_2.IP \\ \wedge Access_1.command = Access_2.command \wedge Access_1.user = update.user \\ \wedge Access_1.object = update.object \wedge update.command = "update" \\ \wedge Between(17, 8) \\ \wedge allowed(Access_1.user, Access_1.object, Access_1.command) \end{array} \right\}$$

\mapsto **action:** *Signal Early Warning (medium)*

4.3.3 Example 3 (Inference)

Insider Breach Scenario: An organisation's acceptable usage policy states that Human Resource employees are authorised to access employee-related objects (O1, O2, O3) separately, but cannot access them combined (i.e. full details of an employee). The insider privacy breach happens when he/she tries to access the objects separately in order to deduce the full details of an employee. The interval between two events is less 60 time units.

PolicyRule_DataInference \triangleq

$$\left\{ \begin{array}{l} \text{Event: } Access_1(user, object, command, session)^3; \\ interval(1, 60); Access_2(user, object, command, session)^3; \\ interval(1, 60); \\ Access_3(User, object, command, Session)^3 \\ \text{Condition: } Access_1.user = Access_2.user \wedge Access_2.user = Access_3.user \wedge Access_3.user = Access_1.user \wedge \\ Access_1.object = "O1" \wedge Access_2.object = "O2" \wedge Access_3.object = "O3" \wedge \\ Access_1.command = "read" \wedge Access_2.command = "read" \wedge Access_3.command = "read" \wedge \\ Access_1.session = Access_2.session \wedge Access_2.session = Access_3.session \wedge \\ Access_3.session = Access_1.session \wedge \\ between(17, 8) \wedge \\ allowed(Access_1.user, Access_1.object, Access_1.command) \\ allowed(Access_2.user, Access_2.object, Access_2.command) \\ allowed(Access_3.user, Access_3.object, Access_3.command) \end{array} \right\}$$

\mapsto **action:** *Signal Early Warning (medium)*

This above scenario states that the user is permitted to perform his activities, as the AEWRC can check the authorisation of the request depending on the outcome of the allowed state (*ready* event) in the pre-check state (see the computational model, Figure 3.2).

Therefore, we make use of the effect of this decision in our language by specifying the predicate *allowed* (*Access₁.user, Access₁.object, Access₁.command*). Upon satisfying the above scenario, this rule will be fired, triggering the medium early warning level *Signal Early Warning (medium)*.

4.3.4 Example 4 (Access for unauthorised purposes)

Insider Breach Scenario 1: a data protection code states that a particular object (e.g. credit card number) can only be accessed for auditing purposes. We assume that the object has been accessed by at least two users with the same role. The privacy breach occurs when the first access request has been performed for the purposes of auditing, followed by an access request for marketing purposes. The duration of the scenario is 8 time units.

PolicyRule_MultipleAccessforData \triangleq

$$\left\{ \begin{array}{l} \text{Event: } (Access_1 (user, role, object, command, session, purpose) \vee \\ \quad Access_2 (user, role, object, read, command, session, purpose)); \\ \quad Access_3 (user, role, object, read, command, session, purpose) \\ \\ \text{Condition: } Access_1.user = "U1" \wedge Access_2.user = "U2" \wedge Access_3.user = "U3" \wedge \\ \quad Access_1.role = Access_2.role \wedge Access_2.role = Access_3.role \wedge Access_3.role = Access_1.role \wedge \\ \quad Access_1.object = "CreditCdNo" \wedge Access_2.object = "CreditCdNo" \wedge Access_3.object = "CreditCdNo" \wedge \\ \quad Access_1.command = "read" \wedge Access_2.command = "read" \wedge Access_3.command = "read" \wedge \\ \quad Access_1.session \neq Access_2.session \wedge Access_2.session \neq Access_3.session \wedge Access_3.session \neq Access_1.session \wedge \\ \quad Access_1.purpose = "auditing" \wedge Access_2.purpose = "auditing" \wedge Access_3.purpose = "marketing" \wedge \\ \quad \quad \quad \text{Between } (9, 17) \wedge \text{duration } (8) \\ \\ \quad \wedge \text{allowed } (Access_1.role, Access_1.object, Access_1.command, Access_1.purpose) \\ \quad \wedge \text{allowed } (Access_2.role, Access_2.object, Access_2.command, Access_2.purpose) \end{array} \right\}$$

\mapsto **action:** *Signal Early Warning (medium)*

Insider Breach Scenario 2: a data protection code states that an object (e.g. credit card number) can only be accessed for the same purpose as that for which that the object was collected. The privacy breach occurs if the object has been accessed by a user for a purpose which is different from that associated with previous accesses by the same user. The duration of this scenario is 10 time units.

PolicyRule_AccessforDifferentPurposes \triangleq

$$\left\{ \begin{array}{l} \text{Event: } \text{Access}_1(\text{user}, \text{object}, \text{command}, \text{purpose})^3; \\ \quad \text{Access}_2(\text{user}, \text{object}, \text{command}, \text{purpose}) \\ \text{Condition: } \text{Access}_1.\text{purpose} \neq \text{Access}_2.\text{purpose} \wedge \text{Access}_1.\text{user} = \text{Access}_2.\text{user} \wedge \\ \quad \text{Access}_1.\text{command} = \text{"read"} \wedge \text{Access}_1.\text{command} = \text{"read"} \wedge \\ \quad \text{Access}_1.\text{object} = \text{"CreditCdNo"} \wedge \text{Access}_2.\text{object} = \text{"CreditCdNo"} \wedge \text{duration (10)} \end{array} \right\}$$

\mapsto **action:** *Signal Early Warning (low)*

4.3.5 Example 5 (Access for a purpose without having a consent)

Insider Breach Scenario: a data protection code states that a particular object (e.g. VIP patient data) can only be accessed if the access is compatible with specific purpose for which the object have been collected and accessed and if the data subject has consented to access. The privacy breach occurs when the object is accessed for the specified purpose, but the user has not been given consent by the data subject.

PolicyRule_VIPSnooping \triangleq

$$\left\{ \begin{array}{l} \text{Event: } (\text{collect}(\text{data subject}, \text{object}, \text{command}, \text{purpose}); \text{Access}(\text{user}, \text{object}, \text{command}, \text{purpose})) \\ \quad \wedge \neg \text{Consent}(\text{data subject}, \text{object}, \text{command}) \\ \text{Condition: } \text{collect}.\text{data subject} = \text{consent}.\text{data subject} \wedge \text{collect}.\text{object} = \text{consent}.\text{object} \\ \quad \wedge \text{consent}.\text{object} = \text{access}.\text{object} \wedge \text{Access}.\text{user} = \text{"clerk"} \wedge \text{Access}.\text{command} = \text{"read"} \wedge \\ \quad \text{collect}.\text{command} = \text{"specify purpose"} \wedge \text{consent}.\text{command} = \text{"give consent"} \\ \quad \wedge \text{collect}.\text{purpose} = \text{Access}.\text{purpose} \end{array} \right\}$$

\mapsto **action:** *Signal Early Warning (high)*

4.4 Summary

This chapter introduced PBSL, which is based on the computational model in Chapter 3. The syntax of the language and its informal descriptions were presented in Section 4.2. Within the language, privacy breach scenarios and the early warning level corresponding to the insider privacy breach and various timing constraints can be specified in a unified manner in the form of policy rules. The main features of PBSL are its expressiveness, simplicity, practicality, and formal semantics. Thus, PBSL provides an expressive specification language that allows actions to be specified together with insider privacy scenarios in an integrated

manner. A variety of insider privacy behaviours can be represented within a comprehensive specification using various event patterns and timing constraints. That ability to express history-based behaviours using a high-level description in our specification language removed the necessity to use the notion of mutable attributes in UCON. This increases the likelihood of early detection of insider breaches with fewer false alarms.

PBSL policy specifications are based on ECA rules. The standard ECA rules have been extended with various constructs, event-supporting patterns, and timing constraints. In our context, events are used to express patterns that contribute to a privacy breach scenario, while conditions are used to express the attributes constraints and timing constraints related to the breach scenario. Actions trigger the corresponding early warning level once a privacy breach scenario has been satisfied. This early warning action is to trigger an enforcement mechanism that will be explained in Chapter 6.

In Section 4.3, some examples of PBSL specifications were provided to demonstrate the use of the language. These included data theft, masquerading, data inference, access for unauthorised purposes, and access without consent.

The formal semantics of the policy language are represented in the next chapter using LTS with the FLTL formalism, where the language contracts can be animated using a support tool. The policy enforcement is discussed in a later chapter, facilitating the concurrent enforcement of policies on AEWS.

Chapter 5: PBSL – Formal Semantics

Objectives

- Give an overview of LTS and FLTL
- Provide the formal semantics of PBSL in LTS and FLTL

5.1 Introduction

To increase confidence in PBSL, it is important to define its formal semantics. This will add rigour to the use of language, describe its constructs unambiguously, allow the animation of our specifications, and verify the properties of the system. However, some existing policy specification languages lack formal semantics, e.g. EPAL [9] and XACML [70]. In security-critical systems that must protect a huge amount of privacy-critical data (e.g. e-government), it is crucial that the policy language has a sound formalism to enable the analysis of system properties such as safety and liveness, and to detect and resolve conflicts between policies. Additionally, the formal basis of a policy language plays a significant role, in association with a particular enforcement mechanism model, in understanding the behaviour of the overall system to ensure compliance.

There are three general approaches to represent the formal semantics of a specification language [43], [74]. These are as follows:

1- Operational approach

The meaning of a construct is interpreted as a series of computational steps of a machine. Thus, this is concerned with how the constructs can be executed. Common examples of operational semantics including process algebra are CSP (Communicating Sequential Processes) [44] and CCS (Calculus of Communicating Systems) [68].

2- Denotational approach

The meaning of a construct is expressed by mathematical objects. Denotational semantics are exemplified by VDM (Vienna Development Method) [50] and the Z notation [93].

3- Property-based approach

The meaning of a construct is specified by giving properties known as assertions. This approach uses axiomatic and algebraic semantics. Axiomatic semantics express partial properties (preconditions and postconditions) of the effect of the constructs. For

example, Larch [40] uses axiomatic semantics. Algebraic semantics use equations related to the properties of the system, such as in CafeOBJ [31].

It has been argued that not all formalisms can be classified as one of these three approaches [43], as the language may belong to a combination of approaches.

The choice of semantics depends on our requirements. In this thesis, we should animate policies, and verify the system properties over the policy specifications supported by an animation and verification technique. In particular, our requirements for the formalism are to be able to express the notions of:

- Events/actions.
- Concurrency of execution.
- Timing.

To animate and analyse the policies using an executable formalism based on the above requirements, it is important for the formalism to be a timed event-based model according to PBSL. This type of semantics needs to be operational in nature and support our expressive language, including event patterns and timing constraints, to enable suitable animation and verification. In addition, transforming from high- to low-level policy rules (namely operational semantics) means the enforcement mechanism can understand the outcome of rules, so the policies can be enforced. For this reason, LTS and FLTL have been chosen as the underlying formalism for both PBSL and the enforcement mechanism models using the LTSA tool. LTS is used to graphically represent the components of PBSL as state machines based on a timed event-based model by LTSA. FLTL is used to specify the policy rules and the enforcement mechanism behaviour to generate the corresponding LTS by LTSA, and to verify the system properties. FLTL and LTS have been preferred over alternatives for the following reasons: (1) concurrency is a primitive construct; (2) to support our time-dependent and event-driven system; (3) the underlying computational model is depicted as a statechart, can be animated in LTS using LTSA; (4) asynchronous FLTL can be used by LTSA to express operational semantics for PBSL specifications and the enforcement model, and verify

both the event-based (namely quantitative temporal) and state-based (namely quantitative temporal) properties in our timed event-based model.

This chapter is organised as follows. In Sections 5.2 and 5.3, we give an overview of LTS and FLTL. The formal semantics of PBSL are given in asynchronous FLTL and the corresponding LTS in Section 5.4.

5.2 Labelled Transitions Systems (LTS)

LTS represents a system as a set of concurrent components, where each component represents a set of states and transitions between these states. Each transition is labelled, and the label denotes an event that is observable by the system [62]. The overall system behaviour is obtained by composing each component of LTS in parallel to form the overall LTS, where the behaviour of all components is interleaved and synchronised on shared events [60].

A timed event-based model using LTS formalism denotes a discrete time model in which time passes by successive ticks of a system clock. Therefore, an explicit tick event is used in LTS to signal the regular ticks of the clock, where each component is synchronised according to the time [62], [59]. Timed LTS models need to be automatically checked by LTSA to ensure that the time is regularly progressing, i.e. it has no deadlocks and the tick event does not stop in any infinite execution [60].

The Finite State Processes (FSP) is a process algebra based input notation for LTSA to specify LTSs [62]. However, process algebra is widely used to model concurrent and distributed systems using the notions of events/actions and processes.

5.3 Fluent Linear Temporal Logic (FLTL)

FLTL is a formalism based on a linear temporal logic to specify state-based and event-based properties on event-based models using the notion of “fluent”. Fluents have been defined by

Miller and Shanahan [69] as “time-varying properties of the world”. A Fluent in FLTL can be either state-based or event-based. State-based Fluents are state predicates which can be either true if a set of initiating events occur, or false if a set of terminating events occur. The initiating and terminating events in the fluent definitions must be disjoint. These events/actions should be related to the events/actions used in the LTS model through LTSA. The definition of a state-based fluent is as follows:

fluent $Fl = \langle Init_{Fl}, Term_{Fl} \rangle$ **initially** $initially_{Fl}$

where

- Fl is the fluent name
- $Init_{Fl}$ is a set of initiating events
- $Term_{Fl}$ is a set of terminating events
- $initially_{Fl}$ is an initial value which can be true or false. It is false by default.

For example, we assume a fluent *FireAlarmWarning* which states that the fire alarm is initially not in the warning state (i.e. *FireAlarmWarning* is initially false). Therefore, the fluent definition for the state-based *FireAlarmWarning* fluent is as follows:

fluent *FireAlarmWarning* = $\langle startFireAlarm, stopFireAlarm \rangle$
initially *False*

This means that when the initiating event *startFireAlarm* occurs, the fluent *FireAlarmWarning* becomes true. When the terminating event *stopFireAlarm* occurs, the fluent *FireAlarmWarning* becomes false.

In contrast, event-based fluents signal the occurrence of an event. The definition of an event-based fluent is as follows:

$$e = (\{e\}, E - \{e\})$$

where

- $\{e\}$ is a set of initiating events
- $E - \{e\}$ is the set of terminating events, and E is the universe of events

For example, the fluent definition for the event-based fluent *startFireAlarm* is as follows:

$$startFireAlarm = (\{startFireAlarm\}, E - \{startFireAlarm\})$$

FLTL formulae can be expressed with logical and temporal operators [62]. The following table lists the logical operators and gives informal descriptions:

Table 5.1 Syntax of *FLTL*: Logical operators

Operator	Description
&&	Conjunction (<i>and</i>)
 	Disjunction (<i>or</i>)
!	Negation (<i>not</i>)
->	Implication
<->	Equivalence

In the following table, the temporal operators of FLTL are presented with informal descriptions:

Table 5.2 Syntax of *FLTL*: Temporal operators

Operator	Description
X	In the next time (next)
$[]$	Always in the future (always)
$\langle \rangle$	At some time in the future (eventually)
U	At some time in the future until (strong until)
W	Indefinitely or at some time in the future until (awaits or weak until)

The following table lists the other constructs and gives informal descriptions:

Table 5.3 Syntax of *FLTL*: Assertion and Ranges

construct	Description
<i>assert</i>	Assertion defines an FLTL property as an FLTL formulae
<i>forall</i>	conjunction of Fluents
<i>exists</i>	disjunction of Fluents

The resulting LTS trace computed from asynchronous FLTL specifications constructs a Büchi automaton [38] using LTSA. To specify FLTL properties of a timed LTS with the aim of enforcing policies and avoiding policy conflicts, the standard FLTL assertions, known as a synchronous or standard temporal logic, must be translated to asynchronous FLTL assertions, known as asynchronous or metric temporal logic [54], [60]. The difference between the two assertions is that synchronous FLTL expresses properties on the sequence of system states observed at a fixed time rate, whereas asynchronous FLTL expresses properties on the sequence of system states observed after the occurrence of an event. In addition,

asynchronous FLTL can formalise bounded timing constraints, whereas synchronous FLTL cannot. This is very suitable to formalise our policies, which include timing constraints. The tick fluent is used with asynchronous FLTL to mark the beginning of each time unit [7], [59], [60]. In [59], the translation from synchronous FLTL to asynchronous FLTL was defined with using an explicit tick as follows:

$$\begin{aligned}
 Tr([] P) &= [] (tick \rightarrow Tr(P)) \\
 Tr(\langle \rangle P) &= \langle \rangle (tick \wedge Tr(P)) \\
 Tr(P \text{ } \mathbf{U} \text{ } Q) &= (tick \rightarrow Tr(P)) \text{ } \mathbf{U} \text{ } (tick \wedge Tr(Q)) \\
 Tr(\mathbf{X} P) &= \mathbf{X}(!tick \text{ } \mathbf{W} \text{ } (tick \wedge Tr(P)))
 \end{aligned}$$

where

- ($Tr: FLTL_{Sync} \rightarrow FLTL_{Async}$)

For example, we assume the synchronous FLTL $[]FireAlarmWarning$ means that $FireAlarmWarning$ holds at each time unit. When translated to asynchronous FLTL, this becomes $[](tick \rightarrow FireAlarmWarning)$, which means $FireAlarmWarning$ holds after the occurrence of an event.

The policy rules, enforcement mechanism behaviour, and system properties can be specified using asynchronous FLTL. The system properties will be expressed in terms of event-based properties for the AEWRS in order to verify the system properties on the policy and enforcement mechanism models. This is achieved in the verification chapter. In terms of the enforcement mechanism model, constraints on the behaviour of the enforcer are defined in the next chapter. We formally enforce the PBSL specifications at run-time based on the underlying computational model. Conflicts between the policy rules can be detected using the LTSA model checker. Having defined the timing constraints, the notion of a bounded time interval [7] is required to reason about policies that include timing constraints. These constraints are usually associated with privacy breach scenarios in our policies. The

following represent the resulting LTS trace semantics of PBSL constructs that are generated from asynchronous FLTL specifications using LTSA.

5.4 Formal Semantics of PBSL

As mentioned in the previous chapter, a PBSL policy rule consists of events, conditions, and actions. In this section, the formal semantics of these components will be represented in timed LTS using FLTL specifications.

5.4.1 Event Patterns

PBSL can express primitive and complex events. The complex events are composed of temporally related events. Therefore, sequences of events, events in parallel, repetition of an event, alternative events, and the absence of an event can all be expressed in PBSL.

In this section, complex events are animated graphically as state machines in timed LTS, representing the order in which events happen. In addition, as our system is time-dependent, our formalism uses an explicit tick event as a standard way to signal the passage of time. Two state-based fluents in FLTL are defined to denote the occurrence of an event since the last time unit as follows:

$$\text{fluent } E1_Occurs = \langle e1, tick \rangle \quad (1)$$

$$\text{fluent } E2_Occurs = \langle e2, tick \rangle \quad (2)$$

where the fluent $E1_Occurs$ (respectively $E2_Occurs$) holds, when $e1$ (respectively $e2$) has occurred, and terminates by the $tick$ event.

Using this tick event, we can specify the time when an event occurs, as well as timing constraints such as the interval between two events (Section 5.4.3). It is important to mention

that policies are enforced using the interleaving concurrency mode that would be observed in our LTS. This concurrent enforcement will be explained in Chapter 6.

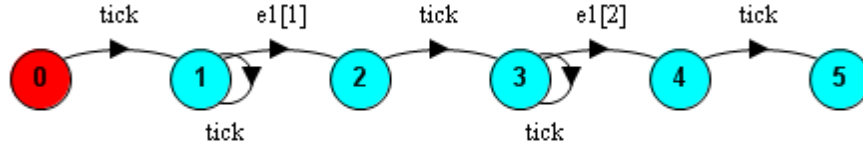
The formal semantics of the basic event constructs and some interesting derived constructs are as follows:

Table 5.4 Formal Semantics of *PBSL*: Event patterns

Name	PBSL
Sequence of events	$e_1 ; e_2 \triangleq \exists t1, t2, t3, t4 (e_1 @ t1) \wedge (tick @ t1) \wedge (tick @ t2) \wedge (e_2 @ t3) \wedge (tick @ t3) \wedge (tick @ t4) \wedge t1 \leq t2 < t3 \leq t4$
LTS (Graphical Representation)	
Name	PBSL
Repetition of an event	$e_1^n \triangleq \exists t1, t2, t3, t4 (e_1^{n-1} @ t1) \wedge (tick @ t1) \wedge (tick @ t2) \wedge (e_1 @ t3) \wedge (tick @ t3) \wedge (tick @ t4) \wedge t1 \leq t2 < t3 \leq t4$

LTS (Graphical Representation)

n=2

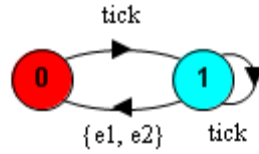


Name	PBSL
-------------	-------------

Events in	$e_1 \parallel e_2 \triangleq \exists t1 (e_1 @ t1) \wedge (e_2 @ t1) \wedge (tick @ t1)$
------------------	---

Parallel	
-----------------	--

LTS (Graphical Representation)

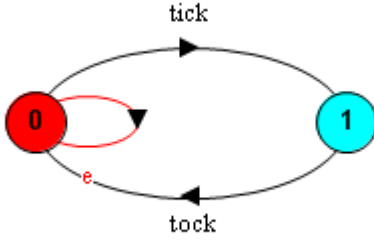


Name	PBSL
-------------	-------------

Alternative events	$e_1 \vee e_2 \triangleq \exists t1, t2 ((e_1 @ t1) \wedge (tick @ t1) \wedge (tick @ t2)) \vee ((e_2 @ t1) \wedge (tick @ t1) \wedge (tick @ t2)) \wedge t1 < t2$
---------------------------	--

LTS (Graphical Representation)



Name	PBSL
Absence of an event	$\neg e \triangleq \exists t0, t1 (\neg (e @ t0) \wedge (tick @ t0) \wedge (tock @ t1) \wedge t0 < t1)$
LTS (Graphical Representation)	
	

5.4.2 Attributes Constraints

In PBSL, attributes constraints are used to constrain the events and the context associated with them. To express the attribute constraints using LTSA, variables need to be defined as parameterised fluents in FLTL, where the variable values are restricted to the range of the fluent parameter. In Table 5.5, the attributes constraints expression for the relationship between a variable and a constant $v \sim x$ is transformed into a fluent-based expression, where a parameterised fluent $v(i)$ records all values of the variable in the range R in which x is evaluated. In the case of the relationship between two variables, the attributes constraints expression $v1 \sim v2$ is transformed into a fluent-based expression in which two parameterised fluents are defined, and the comparison between the fluent parameters is evaluated. The predicate can be captured by a parameterised fluent with a combination of sets of parameters, where all variable values are applied.

Table 5.5 Formal Semantics of *PBSL*: Attribute Constraints

Name	PBSL	FLTL (Textual Representation)
Relationship between a variable and a constant	$v \sim x$ $\sim \in \{<, \leq, >, \geq, =, \diamond\}$	fluent $V [i: R] = <, >$ exists[$i: R$] $V (i) \ \&\& \ (i \sim x)$ $\sim \in \{<, \leq, >, \geq, =, \diamond\}$
Relationship between two variables	$v1 \sim v2$ $\sim \in \{=, \diamond\}$	fluent $V1 [i: R1] = <, >$ fluent $V2 [j: R2] = <, >$ exists[$i: R1$] [$j: R2$] ($V1 (i) \ \&\& \ V2 (j) \ \&\& \ (i \sim j)$) $\sim \in \{=, \diamond\}$
Predicate	$P(v1 \dots vn)$	fluent $P [i: V1] \dots [j: Vn] = <, >$ forall[$i: V1$] ... [$j: Vn$] $P(i, \dots, j)$

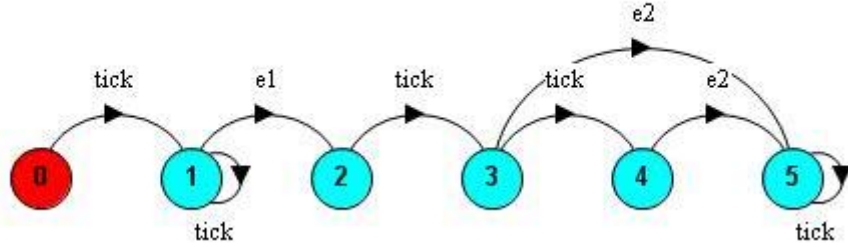
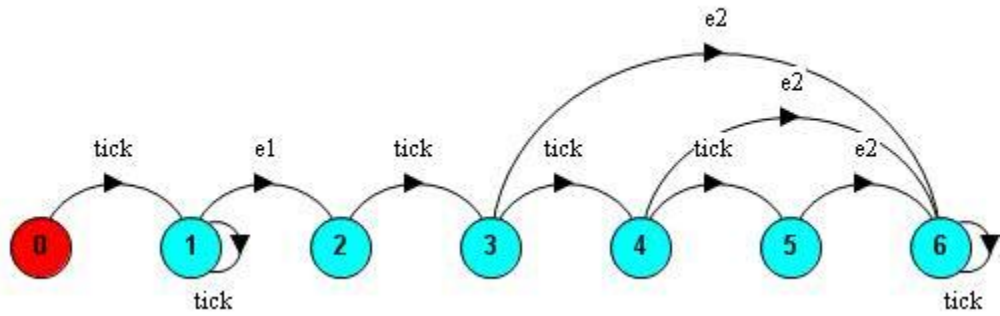
5.4.3 Timing constraints

Timing constraints have been introduced in PBSL to constrain the period of time within which events may occur. These constraints include the time interval (i.e. time period between two events) and the duration of a scenario (i.e. maximum period of time required for the breach scenario to complete). The following describes the semantics of the timing constraints. An explicit tick event is used in our formalism as a standard way of signalling the passage of time. For example, the interval is represented by the minimum and maximum number of

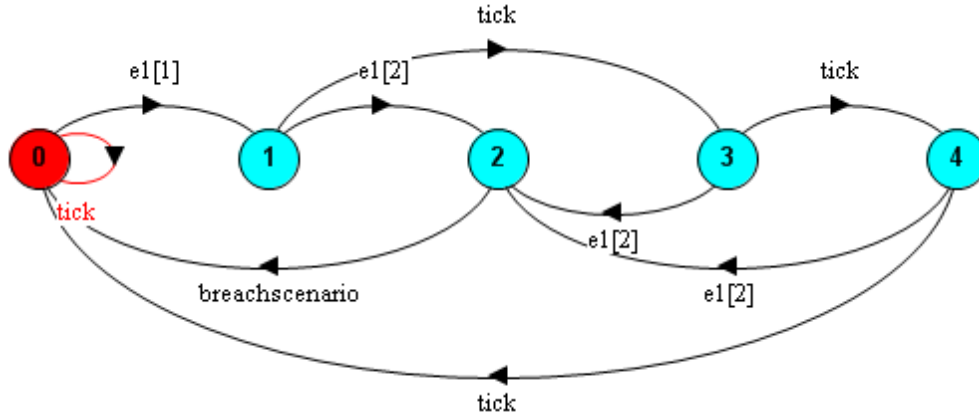
ticks. In the semantics of the interval, the scenario that includes the minimum and maximum number of ticks between the two events occurring is also represented. The formal semantics of some of the interesting timing constraints are as follows:

Table 5.6 Formal Semantics of *PBSL*: Timing Constraints: Interval

Name	PBSL
Interval	$interval(t_{min}, t_{max}) \triangleq \forall interval (tick^{interval} \wedge t_{min} \leq interval \leq t_{max})$
LTS (Graphical Representation)	
<p>$t_{min} = 3, t_{max} = 4$</p>	
Name	PBSL
Interval between two events	$e_1 ; interval(t_{min}, t_{max}) ; e_2 \triangleq \forall interval ((e_1 ; tick^{interval} ; e_2) \wedge t_{min} \leq interval \leq t_{max})$

LTS (Graphical Representation)
 $t_{min} = 1, t_{max} = 2$

 $t_{min} = 1, t_{max} = 3$

Table 5.7 Formal Semantics of *PBSL*: Timing Constraints: duration

Name	PBSL
Duration of a breach scenario	$duration(d)$ $\left\{ \begin{array}{l} \text{Event: } e_1^n \\ \text{Condition: } duration(3) \end{array} \right\} \triangleq \exists t_{start}, t_{end} (e_1 @ t_{start} \wedge e_1^{n-1} @ t_{end} \\ \wedge ((\forall duration(tick^{duration}) \wedge t_{end} = t_{start} + duration \wedge t_{start} \leq \\ duration < t_{end})))$

LTS (Graphical Representation)**d= 3****5.4.4 Early Warning Actions**

As defined in Section 4.2.1.4, the early warning actions of PBSL represent an operation that must be triggered once a privacy breach scenario has been detected. The action determined by the scenario will make the early warning guard be evaluated as true (see computational model, Section 3.3). Once the guard (namely whether the **level** is High, Medium, Low, or Very Low) holds, an interrupt response action must be triggered during the ongoing usage, using guarded enforcement to ensure mutual exclusion. In the formal semantics, the guards are defined as state-based fluents in FLTL, where the fluent can be true when a certain early warning action occurs. These actions are related to the early warning actions used in the LTS model using LTSA. As an outcome of the policy rule, these fluents will be used in the enforcement rules to enforce the policies on AEWS (Chapter 6).

The formal semantics of the early warning actions are as follows:

Table 5.8 Formal Semantics of *PBSL*: Early Warning Actions

Name	PBSL	FLTL (Textual Representation)
Signal Early Warning Level	<i>Signal Early Warning (L)</i> $L \in \text{high, medium, low, vlow}$	$\text{fluent VeryLowEarlyWarning} =$ $\langle \text{signalVeryLowLevel}, \text{tick} \rangle \text{ initially } 1$ $\text{fluent LowEarlyWarning} = \langle \text{signalLowLevel},$ $\text{tick} \rangle$ $\text{fluent MediumEarlyWarning} =$ $\langle \text{signalMediumLevel}, \text{tick} \rangle$ $\text{fluent HighEarlyWarning} = \langle \text{signalHighLevel},$ $\text{tick} \rangle$

5.4.5 Policy Rules

The formal semantics for the components of PBSL have been presented in the previous sections. These components can be combined together to formally represent a policy rule. As defined in Chapter 4, the simple definition of the PBSL policy rule is as follows:

$$\text{PolicyRule} \triangleq \{\text{Privacy breach scenario}\} \mapsto \text{Signal Early Warning (L)}$$

A policy rule specifies the privacy breach scenario and the corresponding early warning level. The implication arrow in PBSL can be captured in FLTL using the operator ($- > X$) or simply ($->$). The operator captures the dependency of the policy decision (namely early warning action) on the previous behaviour.

$$[](\text{tick} \ \&\& \ (\ \text{PrivacyBreachScenario} \ \&\& \ \langle \text{DomPre} \rangle)) \ -> X(! \ \text{tick} \ W \ \text{signal} \langle \text{level} \rangle \text{EarlyWarning})) \quad (1)$$

CHAPTER 5. PBSL FORMAL SEMANTICS

The intuition of the operator is that, whenever *PrivacyBreachScenario* holds on the left, it implies that *signal<level>EarlyWarning* has been triggered, provided that the domain precondition *DomPre* for this early warning action is true. The action of the rule denotes the type of early warning action *<level>* which has been signalled once the privacy breach is satisfied. The domain precondition for early warning action *DomPre* specifies the current state of the early warning level before the early warning action is triggered. The above expression uses a required trigger condition (*ReqTrig*) form in FLTL. This captures an obligation to perform the action [60].

Each component of PBSL can be combined to form a policy rule in asynchronous FLTL. For instance, the formal representation of the policy rule of access for unauthorised purposes is illustrated below. To avoid the problem of state-explosion in LTS using LTSA, we abstract away implementation details with regard to any unnecessary attributes constraints.

```
constraint PolicyRule_AccessforUnauthorisedPurpose =

    Exists [i: Users1] [j: Objects1] [k: Commands1] [l: Purposes1] [m:
Users2] [n: Objects2] [o: Commands2] [p: Purposes2] (tick -> (((Access1
(i,j,k,l)  && Access2 (m,n,o,p)) && ((i == m) && (j == n) && (k == o) &&
(j <> p))) && (X(! tick && ! tick) && (X( ! tick && (X(! tick W
MediumEarlyWarning)))W MediumEarlyWarning))) -> X(!tick W
signalMediumLevel)))
```

(2)

The rule specified above represents a policy to monitor usage to identify access for unauthorised purposes and signal an early warning in such a case. It specifies that the medium early warning signal must be triggered once the same user accesses the same object twice and performs the same command, although the purpose associated with one access is different from that associated with the other access. The duration for the scenario that triggers the rule is 3 time units. When the *signalMediumLevel* action is triggered as the rule fires,

this would make the fluent (state) `MediumEarlyWarning` hold, as defined in Table 5.8 as an initiating action. At the same time, `!MediumEarlyWarning` (the domain precondition for early warning action `signalMediumLevel`) must hold at the next third tick. This domain precondition states that `signalMediumLevel` action must not occur when it is already occurred. Therefore, it specifies the current state of the early warning level if it is not `MediumEarlyWarning`, which must hold before the early warning action `signalMediumLevel` is triggered.

Consequently, the outcomes of the policy rules will influence the state of the AEWRs in using the enforcement mechanisms to enforce the policies and ensuring they comply with the system behaviour. In other words, the policy rules communicate with the enforcement mechanisms that have been modelled in our computational model by triggering the early warning levels. This enforcement mechanism links the states of early warning levels and the actual AEWRs state. Specifying our policies and enforcement mechanism using state machines with LTS facilitates communication between them via events/actions. Thus, being in the early warning level state would immediately enforce the corresponding interrupt response action, changing the AEWRs state, which affects the system state (Section 3.3). For this policy rule to influence the behaviour of the system, it is important to define some rules to link the policy rules and the AEWRs behaviour based on the computational model. Identifying constraints that formally specify the enforcement of policies on AEWRs state within the concurrent settings will be provided in the next chapter.

5.5 Summary

In this chapter, we described the operational formal semantics of PBSL in LTS and FLTL. Asynchronous FLTL is a suitable formalism for specifying the operational semantics of PBSL specifications and verifying event-based properties on our policy model, and is well supported by the animation and verification tool LTSA. Unlike synchronous FLTL, asynchronous FLTL can express timing constraints. Hence, it can be used to reason about the policies that are associated with timing constraints and their enforcements using the LTSA model checker. The graphical representations of PBSL constructs, including event patterns and timing constraints, were presented using timed LTS (traces of events over discrete time

CHAPTER 5. PBSL FORMAL SEMANTICS

steps), which is based on state machines. The resulting LTS trace semantics of PBSL constructs were computed from asynchronous FLTL specifications using LTSA. Finally, we have demonstrated how the formalism can be used to represent policy rules specified in PBSL.

In Chapter 6, an enforcement mechanism will be presented that links the policy rules and the system behaviour based on the computational model. This facilitates the concurrent enforcement of policies on AEWS. Verification of the policy and the enforcement mechanism specifications against the system properties is conducted in Chapter 7.

Chapter 6: Enforcement

Objectives

- Provide operational semantics of the enforcement mechanism model to enforce the PBSL specifications at run-time
- Propose an efficient enforcement mechanism

6.1 Introduction

Having formalised the components of PBSL with timing and event-based characteristics using timed LTS, as generated by asynchronous FLTL with LTSA, in the previous chapter, in this chapter, we describe an enforcement mechanism. The enforcement mechanism is concerned with how policies are enforced in the system. This mechanism causes a mutual interaction between the policy specifications and the AEWRs system behaviour in order to ensure the consistency of policies with the functional behaviour of system. Given that AEWRs is adaptive, time-dependent and security-critical, choosing an efficient enforcement mechanism that adds more rigour in the correctness of the specifications is a crucial task. Some policy rules may have to satisfy certain timing constraints such as the medium early warning must be signalled within 3 time units, or when the medium early warning level holds, access is suspended for a period of 10 time units until some event or a timeout occurs. Therefore, one important requirement is to formalise the enforcement of policies including timing constraints.

Being a UCON-based model, the usage process can be viewed as a long-running access in which the dependency between policy rules is crucial [47]. Conflicts could arise when multiple policy rules are triggered at the same time. For example, conflicts between high and low early warnings could occur when two policy rules are triggered concurrently. Therefore, there is a need to avoid conflicts to ensure system consistency with policies and their enforcement.

In this chapter, we are concerned with the operational formal semantics of the enforcement mechanism as a means of defining enforcement rules and system behaviours, including their interactions to allow for the enforcement of the PBSL policies. The verification of safety and liveness properties against policy and the enforcement mechanism specifications using model checking will be discussed in Chapter 7.

The design of efficient enforcement mechanisms that ensure system compliance against the policies is challenging, particularly when expressive PBSL policies are enforced within a concurrent and distributed setting. We propose an efficient enforcement mechanism

scheme for the concurrent enforcement of policies using the synchronous interleaving model to ensure mutual exclusion. This allows for usage processes to synchronise with time and support condition synchronisation, allowing a capability for interleaving actions to be handled at run-time.

This chapter is organised as follows. The enforcement rules are described in asynchronous FLTL in Section 6.2.1. In Section 6.2.2, we present the operational formal semantics for the system behaviour in asynchronous FLTL. An efficient enforcement mechanism design is presented in Section 6.3.

6.2 Operational semantics of the enforcement mechanism

In this section, we provide the operational formal semantics of the enforcement mechanism for AEWS by defining and specifying the system behaviour and enforcement rules. This enables us to enforce the PBSL specifications at run-time. The policy enforcement is based on the underlying computational model introduced in Chapter 3. Policies can be enforced using four mechanisms: suspension when medium early warning, delay when low early warning, abortion when high early warning and skip when very low early warning. We must avoid policy conflicts, and thus express the mutual exclusion of different policies. These formal semantics are expressed in the form of domain preconditions, domain postconditions, and the required trigger conditions to capture different system states [60]. Having formalised the timing constraints of PBSL in Chapter 5, asynchronous FLTL with the notion of a bounded time interval [7] is used to analyse the behaviour of policies and ensure they satisfy system properties such as safety and progress [62].

The formal semantics of the enforcement rules are expressed in the form of the required trigger conditions, while the formal semantics of the system behaviours are expressed in the form of domain preconditions/domain postconditions and trigger conditions. The semantics will be described in asynchronous FLTL in the next two sections.

6.2.1 Policy Enforcement Behaviour

Having specified the formal semantics of the policy rules in Chapter 5, it is essential to specify their outcomes, which influence the state of the system, in order to formally analyse the policies. This enables the enforcement of PBSL specifications at run-time. The outcome of policy enforcement causes an interaction between the policy specifications and the system behaviour to ensure the consistency of these policies with the AEWRS system behaviour.

To instigate the interaction between the early warning policy specifications and the system behaviour, we specify the enforcement rules on the behaviour of our enforcement mechanism according to the computational model. The outcomes of the early warning policy enforcement by the enforcement rules will change the system state of the AEWRS. It will change from the current state of system, represented by the domain precondition of the interrupt response action, to the next state of the system, represented by the domain postcondition of the interrupt response action. Therefore, the system behaviour depends on the outcome of the policy enforcement, as will be illustrated in the next section.

The early warning action determined by the privacy breach scenario causes the early warning guard to be evaluated as true. Once the guard (namely whether the level is High, Medium, Low, or Very Low) holds, an interrupt response action must be triggered during the ongoing usage, using guarded enforcement to ensure mutual exclusion. In the formal semantics of PBSL early warning actions described in Chapter 5, the guards were defined in FLTL as state-based fluents, where the fluent is true under the occurrence of a certain early warning action. In this section, these fluents will be used as an outcome of the policy rule in the enforcement rules to trigger the interrupt response actions.

To trigger the interrupt response action when the early warning level fluent holds, we use the required trigger condition (*ReqTrig*), which captures an obligation to perform the action [62]. The enforcement rules of the policies are specified in this form. The required trigger condition (*ReqTrig*) takes the following form in FLTL [7], [62]:

CHAPTER 6. ENFORCEMENT

$$\begin{aligned}
 & \text{Constraint ReqTrig_}<\text{ResponseActionName}>_When_<\text{EarlyWarningLevelName}> = \\
 & [] (\text{tick} \rightarrow ((<\text{EarlyWarningLevel}> \ \&\& \ <\text{DomPre}>) \rightarrow X (!\text{tick} \ W \\
 & <\text{InterruptResponseAction}>)))
 \end{aligned} \tag{3}$$

where

- *EarlyWarningLevel* is a state-based fluent representing the early warning level fluent that must be true to trigger the interrupt response action, provided that the domain precondition *DomPre* for this response action is also true.
- *InterruptResponseAction* is an event-based fluent representing the interrupt response action triggered when the *EarlyWarningLevel* and the domain precondition *DomPre* for this response action are true.

Based on the required trigger condition described earlier, the remaining enforcement rules are specified in asynchronous FLTL with their LTS's representations as follows:

$$\begin{aligned}
 & \text{constraint REQ_TRIG_DelayWhenLowLevel} = \\
 & [] (\text{tick} \rightarrow ((\text{LowEarlyWarning} \ \&\& \ !\text{Delayed}) \rightarrow X (!\text{tick} \ W \ \text{delay})))
 \end{aligned} \tag{4}$$

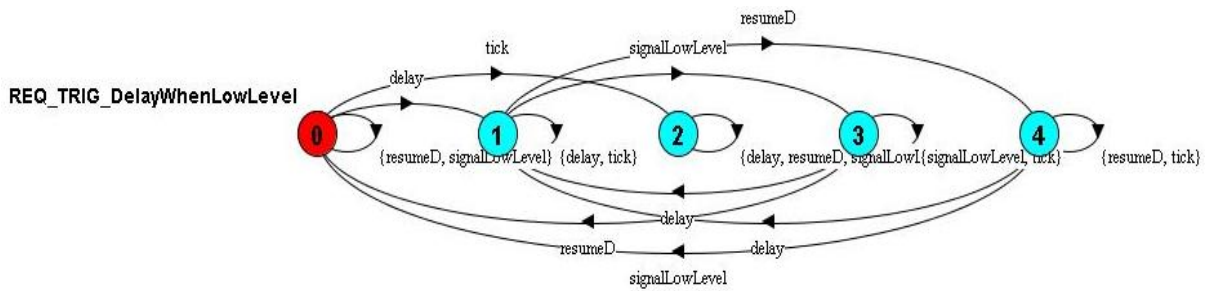


Figure 6.1 LTS representation for *REQ_TRIG_DelayWhenLowLevel*

CHAPTER 6. ENFORCEMENT

constraint REQ_TRIG_SuspendWhenMediumLevel =

$$[](\text{tick} \rightarrow ((\text{MediumEarlyWarning} \ \&\& \ !\text{Suspended}) \rightarrow X(!\text{tick} \ W \ \text{suspend}))) \quad (5)$$

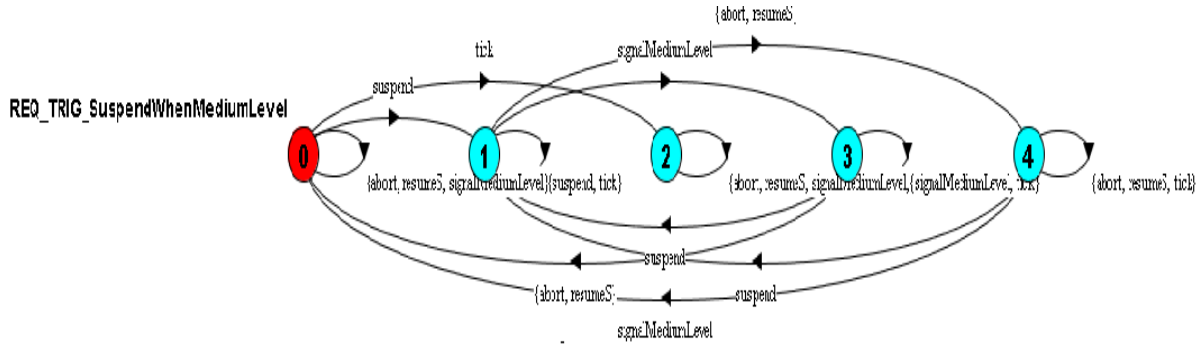


Figure 6.2 LTS representation for *SuspendWhenMediumLevel*

constraint REQ_TRIG_AbortWhenHighLevel =

$$[](\text{tick} \rightarrow ((\text{HighEarlyWarning} \ \&\& \ !\text{Aborted}) \rightarrow X(!\text{tick} \ W \ \text{abort}))) \quad (6)$$

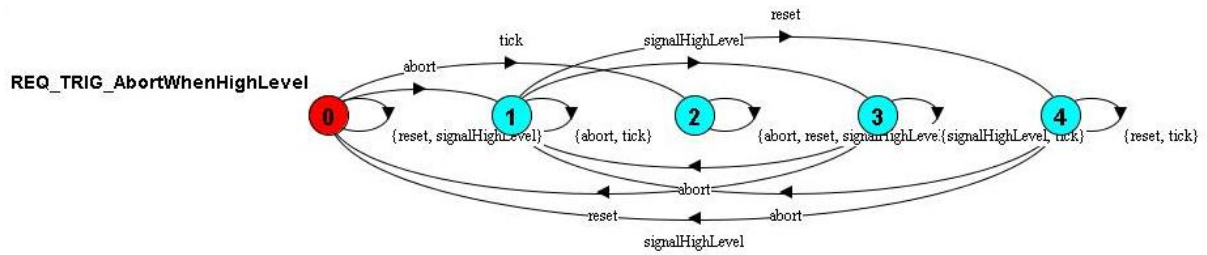


Figure 6.3 LTS representation for *REQ_TRIG_AbortWhenHighLevel*

constraint REQ_TRIG_SkipWhenVeryLowLevel =

$$[](\text{tick} \rightarrow ((\text{VeryLowEarlyWarning}) \rightarrow X(!\text{tick} \ W \ \text{skip}))) \quad (7)$$

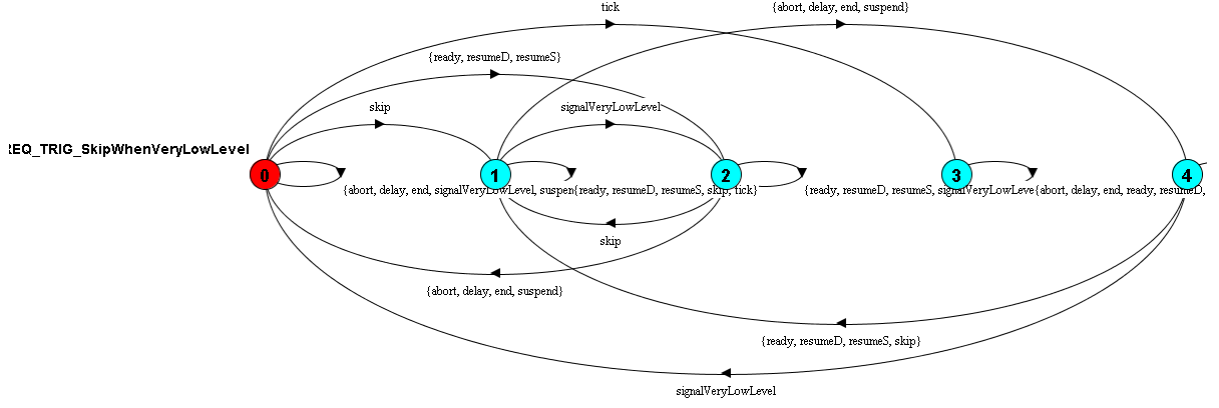


Figure 6.4 LTS representation for *REQ_TRIG_SkipWhenVeryLowLevel*

For example, the constraint *REQ_TRIG_DelayWhenLowLevel* specifies the behaviour of an enforcement rule which triggers the interrupt response action *delay* at the next *tick* if *LowEarlyWarning* and *!Delayed* (the domain precondition for the interrupt response action *delay*) are satisfied. The outcome of this constraint when the *delay* is triggered is reflected in the *Delayed* state, which transitions from not delayed to delayed. The outcome of the rules will be defined and specified in the next section.

6.2.2 AEWRs System Behaviour

In this section, the formal semantics for the system behaviour of the AEWRc are presented. The outcome of the enforcement rules described in the previous section interacts with the system behaviour to ensure compliance with the policy specifications. In other words, the outcomes of enforcing the early warning policies change the AEWRs state, as in the computational model. Therefore, we use asynchronous FLTL to define and specify the states of AEWRc, and the interrupt response actions to be performed to reflect these states.

The system behaviour of the AEWRc is stateful, i.e. it adapts dynamically depending on changes in the system state. The state of the AEWRc will be defined as a state-based fluent in FLTL, where the fluent is true if a certain interrupt response action occurs, or false if another response action occurs. The behaviour of the AEWRc is defined in terms of fluents as follows:

CHAPTER 6. ENFORCEMENT

fluent Suspended = $\langle \text{suspend}, \{\text{resumeS}, \text{abort}\} \rangle$ (8)

fluent Delayed = $\langle \text{delay}, \text{resumeD} \rangle$ (9)

fluent Aborted = $\langle \text{abort}, \text{reset} \rangle$ (10)

fluent NormalOperation = $\langle \{\text{ready}, \text{skip}, \text{resumeD}, \text{resumeS}\}, \{\text{delay}, \text{suspend}, \text{abort}, \text{end}\} \rangle$ (11)

The state-based fluents are defined as: Normal Operation, Delayed, Suspended, and Aborted. Thus, when the initiating response action occurs, the fluent holds, and when the terminating response action occurs, the fluent will cease to hold. For instance, (8) defines the behaviour of the AEWRCS in the Suspended state, which is referred to as a *Suspended* fluent. This fluent will hold (i.e. be true) when the response action *suspend* is initiated, and no terminating action, namely *resumeS* or *abort*, has occurred. The *Suspended* fluent will cease to hold (i.e. be false) when the *resumeS* or *abort* action occurs.

We capture the state of the AEWRCS by formally specifying a domain precondition and domain postcondition for each response action. These conditions capture the elementary state transitions defined by the application of operations in that domain. They capture the transition between the current state and the next state. The domain precondition specifies the current state of the system, which will hold before the action is triggered, while the domain postcondition specifies the state of the system that will hold after the action has been triggered.

To specify a domain precondition (DomPre) and postcondition (DomPost) pair for an interrupt response action, the following form is used [7], [62]:

constraint DomPre__DomPost__InterruptResponseDecision =
 $[] \ (\ (\ \text{tick} \ \&\& \ ! \ \langle \text{DomPre} \rangle \) \ \rightarrow \ X \ (\ ! \langle \text{ResponseAction} \rangle \ W \ \text{tick} \) \)$
 $\&\& \quad [] \ ((\text{tick} \ \&\& \ \langle \text{DomPost} \rangle) \ \rightarrow \ X \ (! \langle \text{ResponseAction} \rangle \ W \ \text{tick}))$ (12)

where

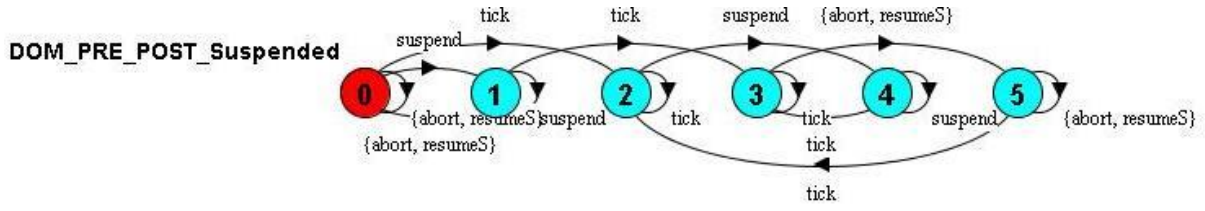
- *ResponseAction* in (12) is an event-based fluent which cannot occur when $!<DomPre>$ is true. *ResponseAction* in (13) is an event-based fluent which cannot occur when the *DomPost* is true.
- *InterruptResponseDecision* is a stateful policy name, i.e. a fluent name in the fluent definition, which can be inserted in place of $<DomPre>$ and $<DomPost>$.

For example, the domain precondition for the “suspend” response action will be a state that is not suspended, while the domain postcondition for this response action will be the suspended state. Therefore, the transition between the domain precondition and domain postcondition states will be labelled by the interrupt response action “suspend”. The domain precondition and the domain postcondition can be formally specified as follows:

```
constraint DOM_PRE_POST_Suspended =
( [] ((tick && Suspended) -> X (! suspend W tick)) &&
  [] ((tick && !Suspended)-> X (! resumeS W tick)) &&
  [] ((tick && !Suspended)-> X (! abort W tick))
)
(13)
```

The domain precondition is specified in (14) using the constraint definition, where the precondition is $!Suspended$ for the response action *suspend*, and the domain postcondition is *Suspended*. If *Suspended* is true, then *suspend* cannot occur, while if $!Suspended$ is true, then *resumeS* and *abort* cannot occur.

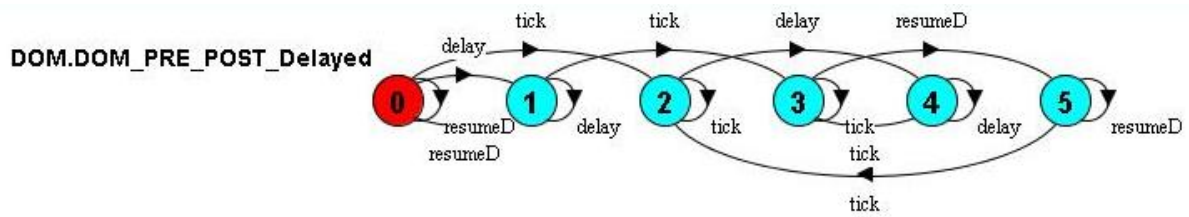
The LTSs representation for the above using the tool LTSA is represented as follows:


 Figure 6.5 LTS representation for *DOM_PRE_POST_Suspended*

The following presents the formal specifications for the domain preconditions and domain postconditions for the remaining response actions of the AEWRC with their LTSs representations:

```
constraint DOM_PRE_POST_Delayed =
( [] ((tick && Delayed) -> X (! delay W tick)) &&
  [] ((tick && !Delayed)-> X (! resumeD W tick))
)
```

(14)


 Figure 6.6 LTS representation for *DOM.DOM_PRE_POST_Delayed*

```
constraint DOM_PRE_POST_Aborted =
( [] ((tick && Aborted) -> X (! abort W tick)) &&
  [] ((tick && !Aborted)-> X (! reset W tick))
)
```

(15)

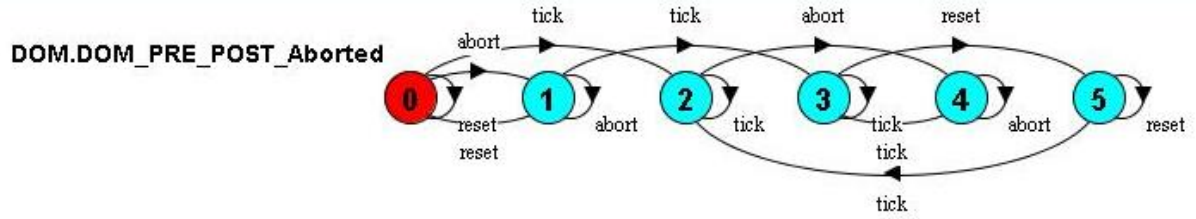


Figure 6.7 LTS representation for *DOM_PRE_POST_Aborted*

```

constraint DOM_PRE_POST_NormalOperation=
(
  [] ((tick && NormalOperation) -> X (! ready W tick)) &&
  [] ((tick && NormalOperation) -> X (! skip W tick)) &&
  [] ((tick && NormalOperation) -> X (! resumeD W tick)) &&
  [] ((tick && NormalOperation) -> X (! resumeS W tick)) &&
  [] ((tick && !NormalOperation) -> X (! end W tick)) &&
  [] ((tick && !NormalOperation) -> X (! delay W tick)) &&
  [] ((tick && !NormalOperation) -> X (! suspend W tick)) &&
  [] ((tick && !NormalOperation) -> X (! abort W tick))
)

```

(16)

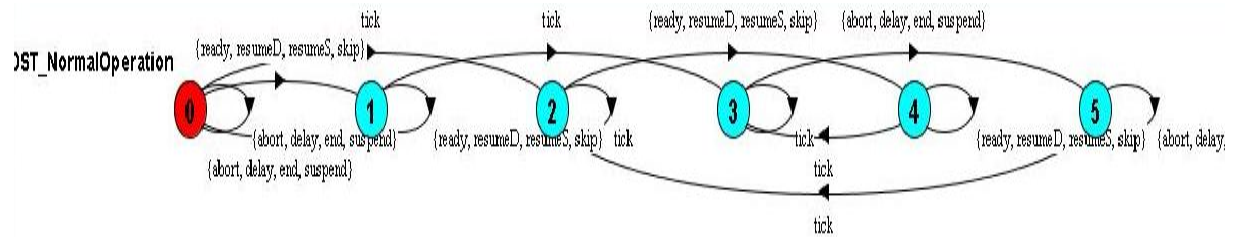


Figure 6.8 LTS representation for *DOM_PRE_POST_NormalOperation*

CHAPTER 6. ENFORCEMENT

Based on the computational model in Section 3.3, the transitions between the AEWRS system states are caused when the response actions are triggered. As defined earlier, the response actions cause the transitions between the system states, making an initiation of some fluents and termination of other fluents. To move from one state to another, we need to specify some obligations to perform the response actions that make transitions between these states. These include moving from the delayed state to the normal operation state, from the suspended state to the normal operation state and from the suspended state to the aborted state. These can be specified using required trigger conditions in which response actions must be performed when the condition and the domain precondition hold or over the passage of time.

The outcome of *delay* interrupt response action when enforced is reflected in the *Delayed* state, within which the enforcement mechanism will observe a number of ticks within this state. On passage of time within the bounded time interval when the delayed state holds, the system must raise *resumeD* event to initiate *NormalOperation* fluent and terminate *Delayed* fluent. An explicit tick event is used in our formalism to signal the passage of time.

To trigger the *resumeD* action when the *Delayed* fluent holds and on passage of three time units (delayed period), we use the following required trigger condition:

```
constraint REQ_TRIG_ResumeWhenDelayedAndTimeout =  
[] (tick -> (( Delayed && !NormalOperation) -> X(! tick && ! tick) && (X(  
! tick && X(! tick W resumeD))) W resumeD))) (17)
```

The constraint in (17) specifies the behaviour of the response action *resumeD* to make the transition from delayed state to the normal operation state. It states that it must be triggered by the next fourth tick if the system state is *Delayed* and not in the normal operation (*!NormalOperation* as a domain precondition for the response actions *resumeD*). The outcome of this constraint when the *resumeD* is triggered is reflected in the next state, where *NormalOperation* fluent is initiated and *Delayed* fluent is terminated. To avoid

deadlocks and time progress violations, we have placed $(! tick \ \&\& \ ! tick)$ in these asynchronous FLTL formulas (17), (18), and (19). For the ease of representation, the maximum period of time to be taken does not exceed 4 time units. The LTS's representation for the constraint (17) using the tool LTSA is represented as follows:

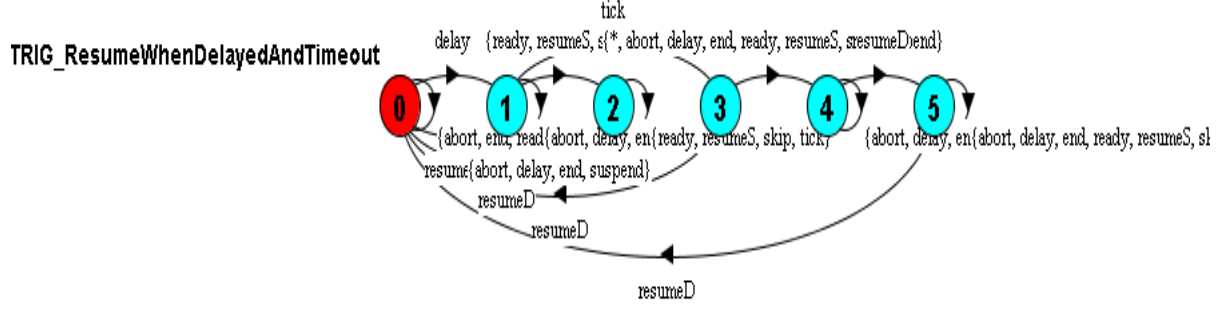


Figure 6.9 LTS representation for $REQ_TRIG_ResumeWhenDelayedAndTimeout$

While the outcome of *suspend* interrupt response action when enforced, is reflected in the *suspended* state, within which the enforcement mechanism will observe a number of ticks until some event occurs. On the occurrence of the tick proceeding the occurrence of the event, the system must raise *resumeS* event to initiate *NormalOperation* fluent and terminate *Suspended* fluent. Otherwise, an *abort* event is generated if no event has occurred within the bounded time interval (suspension period). *Event_Occurs* state-based fluent is defined to denote the occurrence of an event in the last time unit as: *fluent Event_Occurs* = $\langle event, tick \rangle$, where the fluent *Event_Occurs* holds, when *event* has occurred, and is terminated by the *tick* event.

To trigger the *resumeS* action when the *Suspended* fluent holds and on the occurrence of event within the next two time units, we use the following required trigger condition:

```
constraint REQ_TRIG_ResumeWhenSuspendedAndAndEventOccurs =
  [] (tick -> (( Suspended && !NormalOperation ) && (X(! tick && ! tick) &&
    (X( ! tick && (X(! tick W Event_Occurs))) W Event_Occurs))) -> X(! tick W
    (tick && resumeS))))
```

(18)

The constraint in (18) triggers the action *resumeS* at the tick proceeding if the system state is *Suspended* and not in the normal operation (*!NormalOperation* as a domain precondition for the response actions *resumeS*) and the event has occurred (*Event_Occurs*) within the next two ticks. The outcome of this constraint when the *resumeS* is triggered is reflected in the next state, where *NormalOperation* fluent is initiated and *Suspended* fluent is terminated. The LTS's representation for (18) is represented as follows:

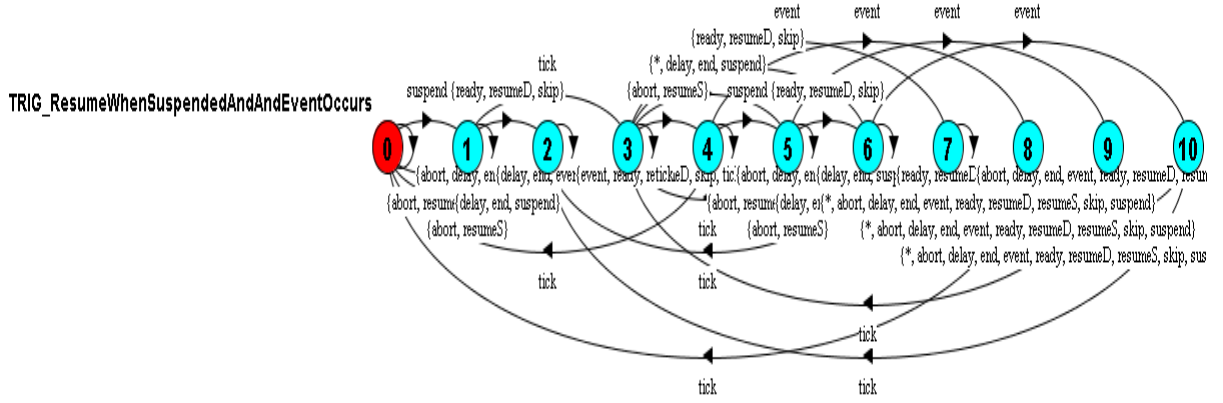


Figure 6.10 LTS representation for *REQ_TRIG_ResumeWhenSuspendedAndAndEventOccurs*

Otherwise, if no event has occurred within the suspension period, we use the following trigger condition to trigger the *abort* action to cause a transition from suspended state to the aborted state:

```
constraint REQ_TRIG_AbortWhenSuspendedAndNoEventAndTimeout =
    [] (tick -> (( Suspended && !Aborted) && (X(! tick && ! tick) && (X( !
    tick && (X(! tick W !Event_Occurs))) W !Event_Occurs))) -> X(! tick W (tick &&
    abort))))
```

(19)

The constraint in (19) triggers the response action *abort* at the fourth tick if the system state is *Suspended* and not in aborted state (*!Aborted* as a domain precondition for

the response actions *abort*) and no event occurs (*!Event_Occurs*) within the next two ticks. The outcome of this constraint when *abort* is triggered is reflected in the next state, where *Aborted* fluent is initiated and *Suspended* fluent is terminated. The LTS's representation for the above using LTSA is represented as follows:

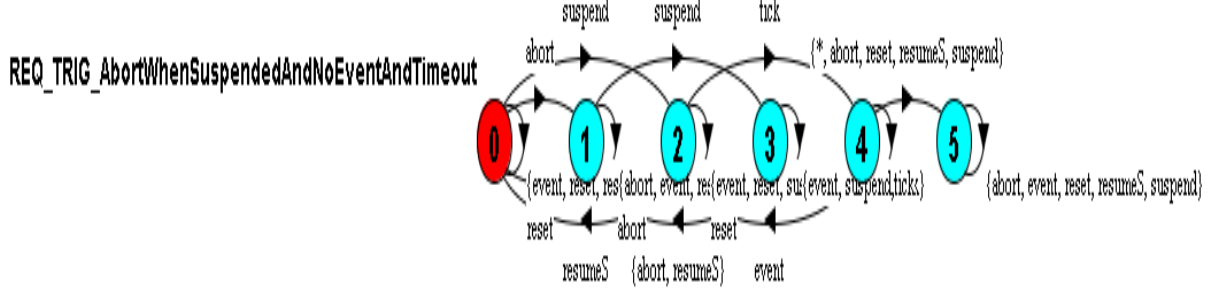


Figure 6.11 LTS representation for *REQ_TRIG_AbortWhenSuspendedAndNoEventAndTimeout*

The operational semantics presented in this section described the interactions between policy rules, enforcement mechanisms, and system behaviour in order to allow for the enforcement of the PBSL policies. We specified the enforcement rules on the behaviour of our enforcement mechanism according to the computational model in Section 3.3. The enforcement rule triggers the interrupt response action if the early warning level fluent and the domain precondition for the interrupt response action are satisfied. The outcome of the early warning policy enforcement will change from the current state of system to the next state. We specified constraints as a domain precondition and domain postcondition for each response action to state that it cannot be triggered if it has already been triggered, and the system states (fluents) cannot be initiated if it is already initiated. In addition, we expressed constraints as trigger condition for each response action that states when the response action must be triggered, provided the precondition is true, to cause the transitions between the system states. This is more intuitive and manageable than using enabling and disabling policies via obligation rules [96], or activation and deactivation rules [48], which are not only difficult to read, but also involve many rules that need to be managed. Further, the policies can be analysed using LTSA, even when the number of states increase, as this tool supports model checking with FLTL. LTSA can check for desirable and undesirable policy properties, and examine the interaction between policy rules, enforcement mechanisms, and system

behaviour. Identifying those operational specifications supports constructing fully operational enforcement mechanism framework.

6.3 Efficient enforcement mechanism design

Janicke et al. [47] proposed that the issue of the concurrent enforcement of UCON policies could be resolved using a static analysis of the dependencies between policy rules. Their approach is based on interleaving enforcement, where only one action can be performed at a time. Concurrent enforcement can be modelled using a true parallelism enforcement model or an interleaving enforcement model. In true parallelism (also known as real concurrency), multiple actions are executed at the same time (simultaneously), while the interleaving model (also known as pseudo-concurrency) is another form of concurrent execution in which multiple actions are interleaved, rather than executed at the same time [62]. We choose to model concurrency using interleaving, because the AEWRC in our model should run on both single- and multi-processor systems. Therefore, we should analyse the policies using tool support to ensure mutual exclusion, whereby only one action can be performed at a time.

There is a range of tool support for the analysis of concurrent systems, such as simulation, theorem proving, and model checking [35]. We selected model checking for the following reasons. The LTSA tool supports model checking using asynchronous FLTL. In addition, asynchronous FLTL can be used by LTSA to specify enforcement rules and AEWRS system behaviour, and to verify event-based properties on our enforcement model. The property violations and traces of events/actions require some interpretation to ensure a correspondence between the enforcement mechanism model behaviour and the real system behaviour.

Our semantics are single point-based in contrast to interval-based [21] where sequence of system states observed at the occurrence of an event at single point of time rather than state-based which are observed at a fixed timed rate. That means that an event occurs at a single point of time. In addition, that would let the enforcement mechanism monitor the system at single point of time rather than continuously. In addition, we use a discrete time model rather than a dense model because the latter introduces additional complexities with regards to model checking while the former widely exists in enforcement mechanism

implementations. Hence, the notion of a bounded time interval is adopted to analyse the behaviour of policies with timing constraints despite the fact we cannot represent the dense or precise time. This enables us to satisfy policy properties such as time progress. The ability to verify that the time is often and eventually progressing by checking a time progress property with the LTSA model checker will be explained in the next chapter.

We employ guarded commands [87] to enforce policies by execution monitoring. Hence, the enforcement mechanism only triggers an interrupt response action when an early warning guard holds. This ensures mutual exclusion, where only one interrupt response action can be triggered at a time. We use condition synchronisation [62] which allows the enforcement mechanism to block processes until the occurrence of a particular action and a specified time period has elapsed. In the next chapter, we use the LTSA model checker to verify that the safety property of mutual exclusion is satisfied, and to detect any possible conflicts between policies (Section 7.3.1).

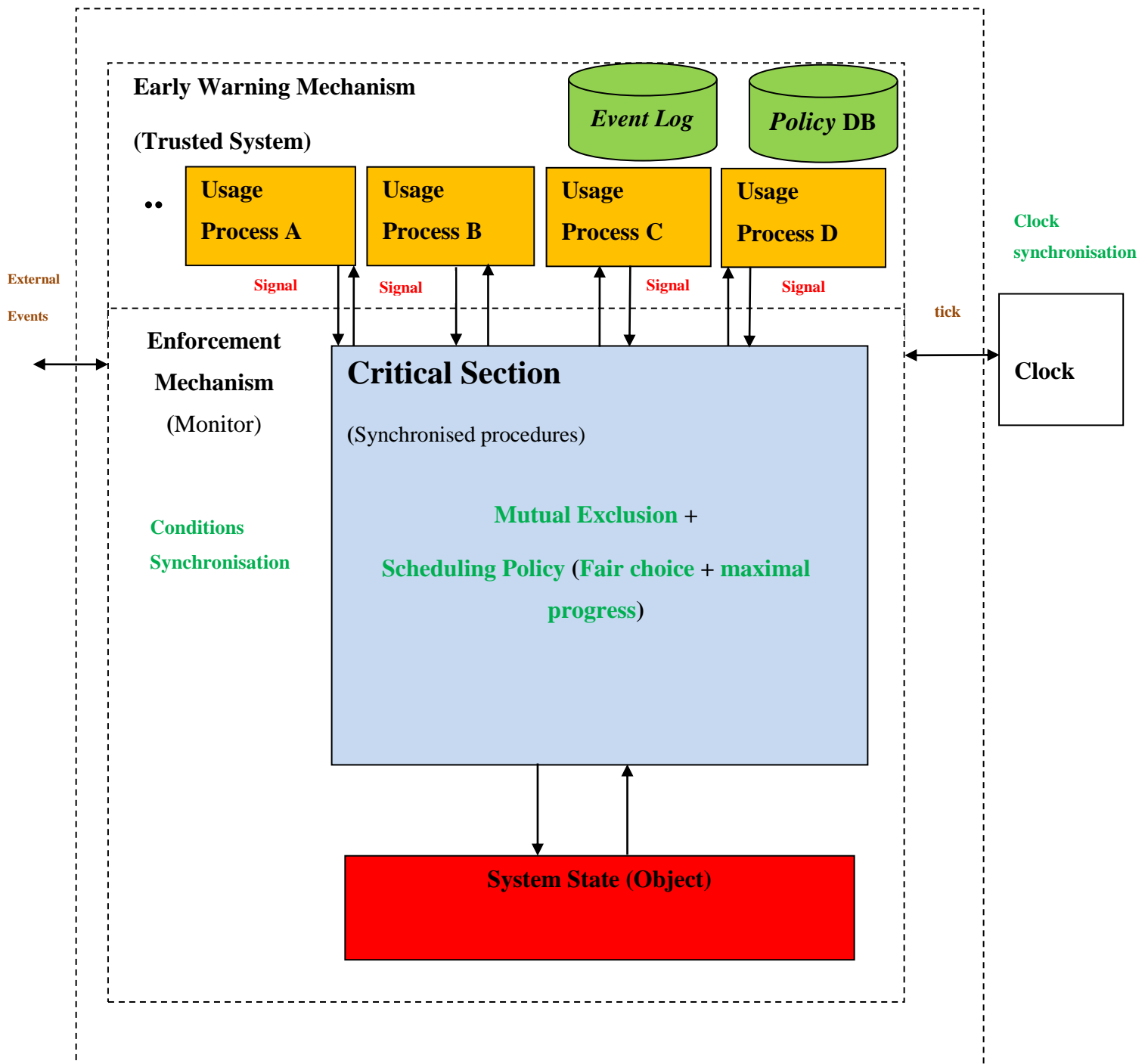


Figure 6.12 Concurrent and synchronous enforcement mechanism for conditions, clock synchronisation, and mutual exclusion

The concurrent enforcement mechanism in Figure 6.12 depicts the concurrent and synchronous enforcement of policies for mutual exclusion and clock and condition synchronisation. The enforcement mechanism acts as a monitor, encapsulating the state of the system. The system state can only be updated by guarded commands, which can be implemented as synchronised procedures. Only a single procedure may be active at a time. Thus, the enforcement procedures ensure mutual exclusion in the update of the affected state of the system, changing the interrupt policy decisions dynamically when an early warning level signals and when a particular time period has elapsed.

The enforcement mechanism ensures that enforcement procedures are synchronised based on time and when a condition holds. This ensures the mutual exclusion for access to the system state. Within the early warning mechanism, whenever time progresses, each process signals a sequence of methods or procedures to the enforcement mechanism corresponding to the early warning level depending on specified policies. The enforcement mechanism updates the state of the shared object by executing synchronised procedures. These synchronised procedures acquire the lock before access the object state and release the lock after access. Therefore, only one process can acquire the lock at a time [62]. The mutual exclusion access to the shared object encapsulating variables prevents the interference problem in which concurrent processes do not update the system state at the same time. The enforcement mechanisms can be implemented as a monitor including the synchronised procedures or the critical section of code belonging to a process; processes that initiate actions can be implemented as active threads.

Considering distributed systems, an important component in the system must be added, that is logging, in which the events are time stamped at local logs based on its own internal clock. They are ordered based on the time at which events occur rather than the order in which events occur. Hence, we assume clock synchronisation [27] to synchronise the process enforcement. For example, when two events occur on different user processes, the timestamp of these events might be different from the order in which events actually occur. Therefore, the clock synchronisation tackles this problem enabling the processes in distributed information systems to be aware of the passage of time and can use the event's timestamps to

specify the events' order and the interval between two consecutive events. However, this scheme results in some synchronisation overhead when ensuring mutual exclusion due to the need to switch between processes. In addition, the synchronous and concurrent enforcement introduces extra complexities with regards to atomicity of events/actions and synchronised procedures executed by the monitor. These particular issues as well as implementation details including the actions and the interfaces are out of scope for this thesis because we leave that to the designer's decision according to their experience and skills and according to the system application domain. Our formalisation allows for verification on the policies and enforcement mechanism models with regards to the safety and progress properties in order to ensure that the system is complete and correct. Thus, that can be used as a sound basis for the concrete implementation of the system with high levels of assurance and understanding.

6.4 Summary

This chapter described the formal operational semantics for the enforcement mechanism of AEWS in LTS and asynchronous FLTL. The resulting LTS semantics of enforcement mechanism were represented as automaton generated from asynchronous FLTL specifications using LTSA. Asynchronous FLTL is a suitable formalism to specify the operational semantics of the PBSL and enforcement mechanism. The system behaviour of AEWS and the enforcement rules have been formally defined as the operational semantics of the enforcement mechanism. The outcome of policy enforcement caused a mutual interaction between the policy specifications and the system behaviour, ensuring the consistency of these policies with the AEWS system behaviour. Formalising PBSL and enforcement mechanism using LTS and FLTL facilitated communication between them. Finally, we proposed a mechanism for the concurrent enforcement of policies using a synchronous interleaving model to ensure mutual exclusion, and supporting clock and condition synchronisation.

In Chapter 7, we evaluate PBSL by demonstrating the use of early warning policies for insider privacy breaches using a case study from the e-government domain. In addition, to complement this evaluation, the safety and time progress properties are verified against the policy and enforcement mechanism models using model checking.

Chapter 7: Verification

Objectives

- Evaluate PBSL language using a case study
- Formal verification of safety and progress properties on the policy and enforcement mechanism models using the LTSA model checker

7.1 Introduction

This chapter describes the evaluation phase of the thesis. The evaluation follows two complementary strands:

- Animation of the enforcement mechanism and policy specifications to check the execution traces of events/actions.
- Formal verification of the AEWRS properties in terms of the enforcement mechanism and policy specifications to check the safety and progress properties.

To facilitate this evaluation, we have chosen a case study from the e- government domain. Our focus in this case study is to demonstrate how early warning policies for insider privacy breaches can be developed to illustrate the verification of satisfaction of the system properties. To evaluate whether the policy specifications with their enforcement mechanism satisfy the AEWRS properties, we formally verify the timed LTS generated from FLTL specifications against the safety and progress properties using the LTSA model checker [62].

We begin by presenting the case study, which involves the examples scenarios of tax revenue system (Section 7.2). This is followed by a formal verification of the satisfaction of the safety and progress properties over the operational semantics of the enforcement mechanism (Section 7.3).

7.2 Case Study

This section provides an overview of an insider-aware tax revenue system (IATRS) (Section 7.2.1), and describes informally the system's acceptable policy usage requirements. Then we give details of the full example scenario of insider breach in this system that is composed of separate scenarios (Section 7.2.2) and illustrates how each of those scenarios can be specified in PBSL (Section 7.2.4).

7.2.1 Insider-aware Tax Revenue System (IATRS)

One example of an e-government system which is in charge of protecting sensitive personal information is the tax revenue system. Tax revenue systems are responsible for collecting and managing the personal information of taxpayers, such as their incomes, addresses, and ages in order to provide services such as benefits, loans, and education. These systems are vulnerable to privacy breaches by insiders in low-level positions such as data entry clerks or those in administrative positions. Privacy breaches are based on some attributes of an insider such as access rights, knowledge of information systems, motivation, intent, and the opportunity to perform their crimes on a particular information asset within a particular time. Privacy breaches could include activities such as the theft of a large amount of citizen data (namely for identity theft), or the browsing of sensitive personal data for the insider's personal gain and the inference of confidential data to which they do not have authorised access for financial gain. The impact of these breaches, in some cases, can have damaging consequences on the organisation such as loss of its reputation, and disruption of operations.

Insiders could pose threats to the organisation individually or in collusion with others. In this case study, we focus on a single insider breaching acceptable usage policies, which is sufficient to illustrate our approach. We leave collusive breach for future work. This case study has specific characteristics that are insider-aware, security-critical, and time-dependent. It is important to note that insider breach scenarios within e-government systems are time-critical, where the insider attack is evolving over time and during a time period. In addition, this case study allows for the evaluation of the expressiveness of PBSL in terms of event history and timing constraints within dynamic environment. The objective of this case study is to demonstrate how simple early warning policy rules for insider privacy breaches can be developed, to illustrate the verification of the satisfaction of the overall system properties.

7.2.2 Scenario Description

The scenario description is derived from [3], [97]. An operator within a tax revenue system is authorised to access sensitive personal taxpayer data stored in a database. The operator's normal working hours are from 08:00–17:00 every weekday; he/she does not usually work outside these hours. The operator's duties include the auditing of income tax, tax returns,

other taxes related to individuals, and their application to various services with regards to benefits, payments, and employment. Other duties include printing letters concerning the collection of accounts and the delivery of benefits with regard to records that have been audited.

7.2.3 The informal system's acceptable usage requirements

- The disclosure of sensitive personal data to a third party is not permitted by any user, either by direct or indirect means.
- The operator is authorised to access citizens' data (namely age, income, address) separately, but is not authorised to view all data related to an individual.
- The user is not authorised to access or print bulk data in the same session.
- The user must not access more than 100 records per day.
- The user must not access any one record more than three times.
- The user must not spend more than 3 min (180 time units) per day on one record.
- The system must only be used within normal working hours (08:00–17:00 every weekday).
- The user is not allowed to access a particular record using two simultaneous connections.

We assume that the operator, Bob, intends to steal sensitive personal data from the tax revenue system and provide it to his wife, Alice, who works as a marketing agent. Therefore, the attacker is located inside the tax revenue organisation. This insider may or may not be authorised to commit the privacy breach.

Bob's intention is to commit data theft. In particular, he tries to steal the phone details of taxpayers, along with their age and income. To do so, Bob has several options. First, he could try to access sensitive personal records frequently, maintaining access for more than the usual time. He could also try to access partial sensitive personal records that are linked to each other by accessing them one by one, thus inferring the full record for a taxpayer. Afterwards, he could try to access a large amount of sensitive personal records in order to get the details

of a group of taxpayers. Finally, Bob could steal the bulk information by printing these sensitive personal data.

A full description of the assumptions for insider breach scenarios are described below:

- 1- Scenario 1 (browsing or snooping):** Within Bob's working hours, we assume he uses his access rights and job role to access partial personal records (age, income, or phone details) three times a day, staying on them for more than the usual time of 3 min (180 time units). This is a potential sign of privacy breach, and might lead to other breaches. The potential breach could happen within different sessions, where Bob logs in to the database 10 times a day and maintains access for more than the usual time (10 min). However, this does not mean that the insider is breaching data privacy regulations, but it will be considered as a suspicious activity.
- 2- Scenario 2 (inference):** The insider breach happens when Bob tries to access interlinked partial data (age, income, and phone data) seven times (referred to as inference variables) in order to deduce the full record. The potential breach could happen within the same session, where the interval between two events is less than 1 min (60 time units). The insider breach happens during working hours.
- 3- Scenario 3 (access to a large amount of data):** Bob accesses a large amount of data in order to obtain phone, age, and income details for taxpayers. The insider breach happens when he makes 100 access requests a day to these details, which is different from his normal behaviour. The breach happens outside working hours.
- 4- Scenario 4 (data theft):** Bob prints the large amount of data he accessed in scenario 3. The insider breach happens during or outside working hours, and the intention is to commit identity theft.

7.2.4 Formalisation in PBSL

In this section, we demonstrate how the four privacy breach scenarios with timing constraints and the early warning level corresponding to the insider privacy breach, which have been specified above, can be expressed—they are expressed in PBSL in the form of four policy rules. The policy rules include high, medium, low, and very low early warning policy rules according to the severity level of the insider breach scenario. The rationale for using some syntactic elements of each policy rule is briefly described.

7.2.4.1 Very Low Early Warning Policy Rule

The policy rule for the first breach scenario (**snooping**) can be specified as follows:

PolicyRule_Snooping \triangleq

$$\left\{ \begin{array}{l} \text{Event: } Access_1(user, object, command)^3 \vee \\ \quad Access_2(user, object, command)^3 \vee Access_3(User, object, command)^3 \\ \\ \text{Condition: } Access_1.user = "Bob" \wedge Access_2.user = "Bob" \wedge Access_3.user = "Bob" \wedge \\ \quad Access_1.object = "age" \wedge Access_2.object = "income" \wedge Access_3.object = "phone" \wedge \\ Access_1.command = "read" \wedge Access_2.command = "read" \wedge Access_3.command = "read" \wedge \\ \quad between(8, 17) \wedge \\ \quad duration(180) \wedge \\ \quad allowed(Access_1.user, Access_1.object, Access_1.command) \wedge \\ \quad allowed(Access_2.user, Access_2.object, Access_2.command) \wedge \\ \quad allowed(Access_3.user, Access_3.object, Access_3.command) \end{array} \right\}$$

\mapsto **Action:** *Signal Early Warning (vlow)*

Where the privacy breach scenario is detected when the event *Access*₁ or (\vee) *Access*₂ or (\vee) *Access*₃ occur 3 times repeatedly with a set of conditions, and timing constraint *duration*(180), leading to the determination of the very low early warning level.

7.2.4.2 Low Early Warning Policy Rule

The policy rule for the second breach scenario (**inference**) can be specified as follows:

PolicyRule_DataInference \triangleq

$$\left\{ \begin{array}{l} \text{Event: } Access_1(user, object, command, session)^7; \\ \quad interval(1, 60); Access_2(user, object, command, session)^7; \\ \quad \quad interval(1, 60); \\ \quad \quad Access_3(User, object, command, Session)^7 \\ \\ \text{Condition: } Access_1.user = "Bob" \wedge Access_2.user = "Bob" \wedge Access_3.user = "Bob" \wedge \\ \quad Access_1.object = "age" \wedge Access_2.object = "income" \wedge Access_3.object = "phone" \wedge \\ Access_1.command = "read" \wedge Access_2.command = "read" \wedge Access_3.command = "read" \wedge \\ \quad Access_1.session = Access_2.session \wedge Access_2.session = Access_3.session \wedge \\ \quad \quad Access_3.session = Access_1.session \wedge \\ \quad \quad \quad between(8, 17) \wedge \\ \quad \quad allowed(Access_1.user, Access_1.object, Access_1.command) \wedge \\ \quad \quad allowed(Access_2.user, Access_2.object, Access_2.command) \wedge \\ \quad \quad allowed(Access_3.user, Access_3.object, Access_3.command) \end{array} \right\}$$

\mapsto **Action:** *Signal Early Warning (low)*

Where the privacy breach scenario is detected when the event $Access_1$, followed by (;) $Access_2$ and followed (;) by $Access_3$ occur 7 times repeatedly, with timing constraints $interval(1, 60)$ between them, triggering the low early warning level action.

7.2.4.3 Medium Early Warning Policy Rule

The policy rule for the third breach scenario (access to a large amount of data) can be specified as follows:

PolicyRule_AccessLargeData \triangleq

$$\left\{ \begin{array}{l} \text{Event: } Access_1(user, object, command)^{100} \wedge \\ \quad Access_2(user, object, command)^{100} \wedge \\ \quad \quad Access_3(user, object, command)^{100} \\ \\ \text{Condition: } Access_1.user = "Bob" \wedge Access_2.user = "Bob" \wedge Access_3.user = "Bob" \wedge \\ \quad Access_1.object = "age" \wedge Access_2.object = "income" \wedge Access_3.object = "phone" \wedge \\ Access_1.command = "read" \wedge Access_2.command = "read" \wedge Access_3.command = "read" \wedge \\ \quad \quad \quad Between(18, 7) \wedge \\ \quad \quad \quad \quad duration(540) \wedge \\ \quad \quad allowed(Access_1.user, Access_1.object, Access_1.command) \wedge \\ \quad \quad allowed(Access_2.user, Access_2.object, Access_2.command) \wedge \\ \quad \quad allowed(Access_3.user, Access_3.object, Access_3.command) \end{array} \right\}$$

\mapsto **Action:** *Signal Early Warning (medium)*

Where the privacy breach scenario is detected when the events $Access_1$ and (\wedge) $Access_2$ and (\wedge) $Access_3$ occur 100 times repeatedly, with timing constraint (540) , leading to the determination of the medium early warning level.

7.2.4.4 High Early Warning Policy Rule

The policy rule for the fourth breach scenario (**data theft**) can be specified as follows:

PolicyRule_DataTheft \triangleq

$$\left\{ \begin{array}{l} \text{Event: } (Access_1(user, object, command, session)^{100}; Print_1(user, object, command, session)^{100}) \wedge \\ (Access_2(user, object, command, session)^{100}; Print_2(user, object, command, session)^{100}) \wedge \\ (Access_3(user, object, command, session)^{100}; Print_3(user, object, command, session)^{100}) \\ \\ \text{Condition: } Access_1.user = Print_1.user \wedge Access_2.user = Print_2.user \wedge Access_3.user = Print_3.user \wedge \\ Access_1.user = "Bob" \wedge Access_2.user = "Bob" \wedge Access_3.user = "Bob" \wedge \\ Access_1.object = Print_1.object \wedge Access_2.object = Print_2.object \wedge Access_3.object = Print_3.object \wedge \\ Access_1.object = "age" \wedge Access_2.object = "income" \wedge Access_3.object = "phone" \wedge \\ Access_1.command = "read" \wedge Access_2.command = "read" \wedge Access_3.command = "read" \wedge \\ Print_1.command = "print" \wedge Print_2.command = "print" \wedge Print_3.command = "print" \wedge \\ Access_1.session = Print_1.session \wedge Access_2.session = Print_2.session \\ \wedge Access_3.session = Print_3.session \wedge \\ duration(1440) \wedge \\ allowed(Access_1.user, Access_1.object, Access_1.command) \wedge \\ allowed(Access_2.user, Access_2.object, Access_2.command) \wedge \\ allowed(Access_3.user, Access_3.object, Access_3.command) \end{array} \right\}$$

\mapsto **Action:** *Signal Early Warning (high)*

Where the privacy breach scenario is detected when the events $Access_1$ followed by $(;) Print_1$ and $(\wedge) Access_2$ followed by $(;) Print_2$ and $(\wedge) Access_3$ followed by $(;) Print_3$ occur 100 times respectively, with timing constraint (1440) , leading to the signalling of the high early warning level action.

It is hard for the insider to circumvent the policies in our system to commit his breaches for the following reasons. First, the system supports multiple enforceable interrupt policies at different risk levels, and can enforce adaptive response actions upon different scenarios of insider threats. Second, the language can explicitly define well-defined policies with expressive insider breach scenario including different access patterns (e.g. sequence, alternative) and timing constraints. Third, the system is capable of detecting attacks at the application level owing to the knowledge of multiple business processes and normal

workflows. It is noteworthy that multiple small steps may not be malicious when those scenarios are considered individually, but can be malicious when all those scenarios are considered in conjunction with each other; therefore, our system monitors all events collectively.

The specification assumes the above-mentioned sequence of the full example scenario with regards to (IATRS). The full scenario was composed of four separate scenarios. We have used of the expressiveness of PBSL to specify the complex policy descriptions with regard to the combining repetitions with some timing constraints or combining different types of constraints. In the following section, we illustrate how early warning actions for the above-mentioned policy rules can be used in the verification process for the safety and progress properties on the entire system.

7.3 Verification

In this section, we use the LTSA model checker [62] to verify the operational semantics of enforcement mechanism against the system properties. In particular, the aim of this verification process is to check the safety and liveness properties of the AEWRS, and that they comply with the enforcement mechanism specifications. The system properties are declarative interrupt policies that are achieved by interrupt policy decisions. Through our formal framework, the enforcement mechanism behaviour should be complete and free from conflicts with regard to these properties. Additionally, the ability to verify the satisfaction of these properties helps us check the consistency between these high-level properties with low-level the enforcement mechanism models.

Given a timed LTS model and a specification of the properties that are required to hold, the LTSA model checker verifies whether the properties asserted are satisfied by the model. Upon detection of a violation, a counterexample is generated. Counterexamples are a sequence of execution traces which violate the property i.e. cause the property to be false. Absence of Counterexamples indicates that the timed LTS model satisfies the property, i.e. cause the property to be true.

Two categories of properties, safety and liveness, were introduced by Lamport [56] to ensure the correctness of concurrent programs. The safety property asserts that nothing bad happens, while the liveness property asserts that something good eventually happens. In the previous chapter, the enforcement mechanism ensures mutual exclusion. Hence, we now verify that the enforcement mechanism actually ensure mutual exclusion, and detect any conflicts by checking the safety property (Section 7.3.1). In terms of the liveness property, we check a particular class of liveness known as the progress property [6], [62]. The progress property asserts that a particular event/action is often and eventually executed. As our formalism is timed using tick events, we verify that the time is often and eventually progressing by checking, in particular, the progress property of time (Section 7.3.2). The animation and verification process was executed on a desktop computer (Core2duo CPU, 2.96 GB of RAM).

7.3.1 Safety Properties

In this section, we need to verify that safety properties for mutual exclusion are satisfied. We assert that no conflicts (violation) can occur between two or more policies when policy rules fire early warning actions at the same time. That is included with safety properties for interrupt policies given time-critical nature (with timing constraints) of the AEWS. The property violations and traces of events/actions require some interpretation by the designer to ensure a consistency between the animated model behaviour and the system properties. This is needed to understand the situations of violations and the potential impact on the real system behaviour. We investigate safety properties and make some modifications to satisfy that two or more interrupt policies must not be true at the same time. Safety properties are specified in asynchronous FLTL to be checked by the LTSA model checker.

Each system property will be expressed as an asynchronous FLTL safety property using the tick fluent [7]. There are four goal patterns to express the property: achieve, cease, maintain, and avoid [30]. We express our properties using a bounded achieve expression [7], [58] with bounded temporal operator such as $\langle \rangle_{\leq d}$ which means within the next d time units in the future. To make the investigation of the violation traces feasible, we do not specify time periods that take more than 4 time units.

Property 1 (safety): *Normal Operation when Very Low Early Warning*

The first property we assert is that “once the guard very low Early Warning Level holds, ongoing usage must be skipped to the normal operation mode by the next time unit”. We express this safety property as follows:

```
assert NormalOperationWhenVeryLowLevel
= [] (tick -> ((VeryLowEarlyWarning) -> X (!tick W (tick &&
NormalOperation))))
```

The above-mentioned specification states that, it is always the case that if the very low early warning holds at the occurrence of a tick, then ongoing usage must be skipped to the normal operation mode by the next tick. Analysing this property on the timed LTS generated using LTSA produces no safety property violation as shown in the following result:

```
Analysing...
Depth 23 -- States: 4975 Transitions: 33140 Memory used: 23748K
No deadlocks/errors
Analysed in: 16ms
```

Property 2 (safety): *Delay when Low Early Warning*

The second property we assert is that “once the guard Low Early Warning Level holds, ongoing usage must be delayed by the next time unit”. We express this safety property as follows:

```
assert DelayedWhenLowLevel
= [] (tick -> ((LowEarlyWarning && !Delayed) -> X (!tick W (tick &&
Delayed))))
```

The above *DelayedWhenLowLevel* assertion states that it is always the case that if the low early warning holds and delayed state does not hold at the occurrence of a tick, then the

CHAPTER 7. VERIFICATION

ongoing usage must be delayed by the next tick. Checking this property using LTSA verifies that *DelayedWhenLowLevel* safety property is not violated as follows:

```
Analysing...
Depth 22 -- States: 4263 Transitions: 28974 Memory used: 23118K
No deadlocks/errors
Analysed in: 15ms
```

Property 3 (safety): *Resumption when delayed and Timeout*

The third property to assert is that “once the usage is delayed, the access must be resumed after three time units i.e. at the time unit that occurs after the occurrence of the third time unit”. We express the required safety property as follows:

```
assert ResumedWhenDelayedAndTimeout
= [] (tick -> ((Delayed && !NormalOperation) -> X(! tick && ! tick) &&
(X( ! tick && X(! tick W (tick && NormalOperation)))) W (tick &&
NormalOperation))))
```

The above *ResumedWhenDelayedAndTimeout* property states that it is always the case that if the *Delayed* holds and normal operation state does not hold (*!NormalOperation*) at the occurrence of a tick, then the ongoing usage must be resumed before the occurrence of the third tick. The expression on the left `[] (tick -> ((Delayed && !NormalOperation) -> X(! tick && ! tick) && X(! tick && X(! tick W (tick && NormalOperation)))) W (tick && NormalOperation))` avoids the situation in which the normal operation and delayed states are true at the same time as that violates mutual exclusion.

Performing the LTL property check produces the following violation trace:

CHAPTER 7. VERIFICATION

```
Trace to property violation in ResumedWhenDelayedAndtimeout:
    tick
    delay          Delayed
    tick           Delayed
    tick           Delayed
    resumed        NormalOperation
    tick           NormalOperation
Analysed in: 0ms
```

The above violation trace indicates that a sequence of traces which violate the *ResumedWhenDelayedAndTimeout* property. The name of fluents on the left are true if the events/actions on the right occur as the LTSA annotates. We notice that at the occurrence of the second tick proceeding the *delay* action, the *Delayed* fluent is true and the *NormalOperation* fluent is false, and then the initiating/terminating action *resumed* occurs afterward. Hence, at the third tick proceeding the *delay* action, the fluent *Delayed* is evaluated to false (when it must have been true) and the fluent *NormalOperation* is evaluated to true (when it must have been false). To avoid the situation where *resumed* occur before the third tick within the delayed state, we modify the above FLTL safety property as the following formulae:

```
assert ResumedWhenDelayedAndTimeout
= [] (tick -> ((Delayed && !NormalOperation) -> X(! tick && ! tick) &&
(X( ! tick && X(! tick W NormalOperation))) W NormalOperation)))
```

This revised property states that it is always the case that if the delayed holds and normal operation state does not hold at the occurrence of a tick, then the ongoing usage must be resumed at the occurrence of the fourth tick.

Checking the revised property on the timed LTS generated using LTSA produces the following result, where there are no safety violations:

CHAPTER 7. VERIFICATION

```
Analysing...
Depth 22 -- States: 4039 Transitions: 27414 Memory used: 21632K
No deadlocks/errors
Analysed in: 15ms
```

Property 4 (safety): *Suspension when Medium Early Warning*

The fourth property that we assert is that “once the guard medium Early Warning Level holds, ongoing usage must be suspended by the next time unit”. We express this safety property as follows:

```
assert SuspendedWhenMediumLevel
= [] (tick -> (( MediumEarlyWarning && !Suspended) -> X (!tick W (tick &&
Suspended)))))
```

The above *SuspendedWhenMediumLevel* assertion states that it is always the case that if the medium early warning holds and suspended state does not hold at the occurrence of a tick, then ongoing usage must be suspended by the next tick.

Running LTL property check on the timed LTS generated by LTSA produces the following result, where there are no safety violations:

```
Analysing...
Depth 22 -- States: 4039 Transitions: 27414 Memory used: 22635K
No deadlocks/errors
Analysed in: 15ms
```

Property 5 (safety): *Resumption when Suspended and an event occurs*

The fifth property to assert is that “once the usage is suspended and an event has occurred within the next two time units, the access must be resumed by the next time unit”. The required safety property is expressed as follows:

CHAPTER 7. VERIFICATION

```
assert ResumedWhenSuspendedAndEventOccurs
= [] (tick -> ((Suspended && !NormalOperation) && (X(! tick && ! tick)
&& (X( ! tick && (X(! tick W Event_Occurs)))W Event_Occurs))) -> X(!tick W
(tick && NormalOperation))))
```

This *ResumedWhenSuspendedAndEventOccurs* property states that it is always the case that if the *Suspended* holds and normal operation state does not hold (*!NormalOperation*) at the occurrence of a tick, and an event has occurred (*Event_Occurs*) at the second and third time units, then the ongoing usage must be resumed by the fourth time unit.

Performing property checking produces the following violation trace:

```
Trace to property violation in ResumedWhenSuspendedAndEventOccurs:
    tick
    suspend      Suspended
    tick         Suspended
    event        Suspended && Event_Occurs
    tick         Suspended
    tick         Suspended
Analysed in: 15ms
```

The above output shows a counterexample which violates the *ResumedWhenSuspendedAndEventOccurs* property. The violation trace is illustrated by the LTSA with the names of fluents that are true on the left if the events/actions on the right occur. We observe in the trace that at the occurrence of the third tick proceeding the occurrence of the satisfying *event*, the *Suspended* fluent is true and *NormalOperation* fluent is false, but the *Suspended* fluent has not been terminated by the action *resumes* afterward. Hence, at the last tick, *NormalOperation* fluent is evaluated to false and not made true by the occurrence of the initiating action *resumes* when it must have been true. To avoid the violation, the required change to the above asynchronous FLTL formulae is to put the

CHAPTER 7. VERIFICATION

operator (\parallel) – disjunction (or) - in place of ($\&\&$) -Conjunction (*and*)-. The revised FLTL safety property is as follows:

```
assert ResumedWhenSuspendedAndEventOccurs
= [] (tick -> ((Suspended && !NormalOperation) && (X(! tick || ! tick)
|| (X( ! tick || (X(! tick W Event_Occurs)))W Event_Occurs))) -> X(!tick W
(tick && NormalOperation))))
```

This modified property states that it is always the case that if the *Suspended* holds and normal operation state does not hold (*!NormalOperation*) at the occurrence of a tick, and an event has occurred (*Event_Occurs*) within the next two time units, then the access must be resumed by the next time unit.

Checking this modified property on the timed LTS generated by LTSA, yields the following output, where there are no safety violations:

```
Analysing...
Depth 22 -- States: 4039 Transitions: 27414 Memory used: 24572K
No deadlocks/errors
Analysed in: 15ms
```

Property 6 (safety): *Abortion when Suspended and timeout and an event has not occurred*

The sixth property to assert is that “once the usage is suspended and no event has occurred within the next two time units, the ongoing usage must be aborted at the fourth time unit”.

We specify the required safety property as follows:

CHAPTER 7. VERIFICATION

```
assert AbortedWhenSuspendedAndNoeventAndTimeout
= [] (tick -> ((Suspended && !Aborted) && (X(! tick || ! tick) || (X(!
tick || (X(! tick W !Event_Occurs))) W !Event_Occurs))) -> X(!tick W (tick
&& Aborted))))
```

The above *AbortedWhenSuspendedAndNoeventAndTimeout* property states that it is always the case that if the *Suspended* holds and aborted state does not hold (*!Aborted*) at the occurrence of a tick, and no event has occurred (*!Event_Occurs*) within the next two time units, then the ongoing usage must be aborted at the next time unit.

Performing LTL property check on the timed LTS generated using LTSA produces the following violation trace:

```
Trace to property violation in AbortedWhenSuspendedAndNoeventAndTimeout:
    tick
    suspend      Suspended
    tick         Suspended
    tick         Suspended
Analysed in: 0ms
```

The above trace shows a clear violation of the *AbortedWhenSuspendedAndNoeventAndTimeout* property. It is clear that at the occurrence of the last tick in the trace the *Suspended* fluent is true and *Aborted* fluent is false, and no event has occurred but has not been finished by the action *abort* to initiate *Aborted* fluent and terminate the *Suspended* fluent. To eliminate this violation, the above asynchronous FLTL formulae needs to be modified by replacing the operator (*||*) to (*&&*) as follows:

```
assert AbortedWhenSuspendedAndNoeventAndTimeout
= [] (tick -> ((Suspended && !Aborted) && (X(! tick && ! tick) && (X(!
tick && (X(! tick W !Event_Occurs))) W !Event_Occurs))) -> X(!tick W (tick
&& Aborted))))
```

CHAPTER 7. VERIFICATION

This revised property states that it is always the case that if the *Suspended* holds and aborted state does not hold (*!Aborted*) at the occurrence of a tick, and no event has occurred (*!Event_Occurs*) at the second and the third tick, then the ongoing usage must be aborted at the fourth tick. Running against this revised property results in no violations as shown in the following output:

```
Analysing...
Depth 22 -- States: 4039 Transitions: 27414 Memory used: 24832K
No deadlocks/errors
Analysed in: 31ms
```

Property 7 (safety): *Abortion when High Early Warning*

The seventh property we assert is that “once the guard high Early Warning Level holds, ongoing usage must be aborted by the next time unit”. The safety property is expressed as follows:

```
assert AbortedWhenHighLevel
= [] (tick -> ((HighEarlyWarning && !Aborted) -> X (!tick W (tick &&
Aborted))))
```

The above specifications state that, it is always the case that if the high early warning holds at the occurrence of a tick, then the ongoing usage must be aborted by the next tick. Checking this property on the timed LTS generated using LTSA generates no violations as is seen the following output:

```
Analysing...
Depth 22 -- States: 4119 Transitions: 27934 Memory used: 23262K
No deadlocks/errors
Analysed in: 31ms
```

7.3.2 Progress Properties

As our LTS formalism is timed using tick events, it is essential that the time is eventually progressing. The trace in which the time does not progress is called Zeno execution [8]. Hence, we need to check that the tick events happen regularly and will be executed infinitely often [60]. This is achieved by model checking the LTS model against a property called time progress [62]. This property is used with the discrete-time event-based model.

Our checking of this time progress assume maximal progress [62], [64] and fair choice [62]. The maximal progress of actions means that all actions that are ready must happen between tick events. Therefore, the system will perform all actions that can occur before the next tick. This is determined by our scheduling policy which gives tick events low priority [62]. That ensures that interrupt policies with timing constraints are enforceable, by giving the events/actions a tick low priority. Subsequently, this allows the events/actions to occur when the components of the system are ready to perform them within a time period. The maximal progress assumption is included in the overall system behaviour. The overall system behaviour is represented by the parallel compositions of all timed LTSs generated from the FLTL operational semantics in Section 6.2. We have included maximal progress by assigning tick events an action low priority using the FSP low priority operator as $>>\{tick\}$. This will ensure that all events/actions that can happen will be performed by the system during a single time unit.

To avoid an infinite number of events/actions happening during a (finite) interval of time (it is called finite variability [64]), we assume that our scheduling policy makes a fair choice. Fair choice assumption defined as follows: *“if a set of transitions are executed infinitely often, then every transition in the set will be executed infinitely often”*. Hence, the enforcement mechanism will eventually observe a tick event whenever the time progresses by a single time unit. We check that the resulting timed LTS of the overall system behaviour satisfy the time progress property where the tick events must happen infinitely often in every infinite execution as follows:

CHAPTER 7. VERIFICATION

```
Progress TimeProgressProperty = {tick}.
```

The **TimeProgressProperty** is specified in FSP that checks the time progress property violations that can happen in all LTS traces generated from the operational semantics of the enforcement mechanism. Performing **TimeProgressProperty** property check on the overall system behaviour in timed LTS, under the maximal progress assumption using LTSA produces the following violation trace:

```
Progress violation: TimeProgressProperty
Trace to terminal set of states:
    signalLowLevel
    tick
    delay
Cycle in terminal set:
    signalLowLevel
Actions in terminal set:
    signalLowLevel
Progress Check in: 0ms
```

The above-mentioned counterexample satisfies the safety property *DelayedWhenLowLevel*, but violates the time progresses property (*TimeProgressProperty*). This leads to the terminal set in which the only *signalLowLevel* action and no tick occurs afterwards. The time progress property is violated where the second tick does not occur after the delay action. This is because there is no domain precondition for the action *signalLowLevel* that states when the *LowEarlyWarning* fluent is initiated; it cannot be initiated by the occurrence of the action *signalLowLevel* again before the next time point, would prevent the low early warning from happening again. Hence, we need to improve the operational semantics with domain preconditions for the action *signalLowLevel* to avoid this progress violation. Additionally, there is a need to define domain preconditions for the other early warning actions to prevent this situation from occurring in those situations as well. The domain preconditions for early warning actions can be formally specified in a similar manner to that for interrupt response actions in (Section 6.2.2), and they are given as follows:

CHAPTER 7. VERIFICATION

```
constraint DOM_PRE_SignalLowLevelWhenLowEarlyWarning =  
    [] (tick -> ((LowEarlyWarning) -> X( !signalLowLevel W tick)))  
  
constraint DOM_PRE_SignalMediumLevelWhenMediumEarlyWarning =  
    [] (tick -> ((MediumEarlyWarning) -> X( !signalMediumLevel W tick)))  
  
constraint DOM_PRE_SignalHighLevelWhenhighEarlyWarning =  
    [] (tick -> ((HighEarlyWarning) -> X( !signalHighLevel W tick)))  
  
constraint DOM_PRE_SignalVeryLowLevelWhenVeryLowEarlyWarning =  
    [] (tick -> ((VeryLowEarlyWarning) -> X( !signalVeryLowLevel W  
tick)))
```

For instance, the domain precondition that the “*signalLowLevel*” action may not be signalled if low early warning is already signalled, as specified in the above-mentioned constraint *DOM_PRE_SignalLowLevelWhenLowEarlyWarning*. Additionally, these specifications ensure that the policy rules triggering early warning actions will comply with the domain precondition for the same actions.

We further analyse the progress of the overall system behaviour against the time progress property (*TimeProgressProperty*). The following progress violation is obtained:

```
Progress violation: TimeProgressProperty  
Trace to terminal set of states:  
    signalMediumLevel  
    tick  
    suspend  
Cycle in terminal set:  
Actions in terminal set:  
    {}  
Progress Check in: 0ms
```

CHAPTER 7. VERIFICATION

This trace satisfies the safety property *SuspendedWhenMediumLevel*, but violates the expectation that the time progresses (*TimeProgressProperty*). The dead lock occurs because no tick occurs in the trace after the first tick. This violation indicates an inconsistency problem in the operational semantics of the enforcement rules. Hence, we need to improve the operational semantics with a required precondition for the *suspend* action to be consistent with the trigger condition for this response action [7], [60]. The required precondition (*Req_PRE*) captures a permission to perform an action. In other words, it captures conditions on the actions that a system component may perform when the condition is met to satisfy the properties. The required precondition for the action *suspend* is specified as follows:

```
constraint REQ_PRE_SuspendWhenMediumLevel =  
    [] (tick -> (!MediumEarlyWarning -> X (!suspend W tick)))
```

The above expression states that the ongoing usage may not be suspended if the medium early warning has not been signalled. To satisfy the time progress property for the system, the required preconditions for each response action need to be added in a manner similar to the one mentioned above to ensure consistency with the required trigger for those response actions expressed in (Section 6.2). In the following, we include some of the interesting required preconditions:

```
constraint REQ_PRE_DelayWhenLowLevel =  
    [] (tick -> (LowEarlyWarning -> X( !delay W tick)))
```

```
constraint REQ_PRE_SkipWhenVeryLowLevel =  
    [] (tick -> (VeryLowEarlyWarning -> X (!skip W tick)))
```

```
constraint REQ_PRE_AbortWhenHighLevel =  
    [] (tick -> (HighEarlyWarning -> X (!abort W tick)))
```

Further progress analysis after adding the missing required preconditions for the rest of actions generate no violation, thus satisfying the time progress property.

```
No progress violations detected.  
Progress Check in: 0ms
```

7.4 Summary

In this chapter, using a case study of a tax revenue system, we illustrated how the full examples scenario of insider breach in this system are identified and formalised from a given natural language. We showed the benefits of our specification approach that utilise its expressiveness to specify some complex policies. We demonstrated how the progress and safety properties of the system with respects to policy and enforcement mechanism specifications can be verified using the LTSA model checker. We expressed safety properties for mutual exclusion using as an asynchronous FLTL property. We eliminated the safety property violations by making some modifications to those specified properties to ensure the consistency between the animated model behaviour and the system properties. We checked the time progress property to ensure that the time is often and eventually progressing. Our checks of the time progress property assumed the maximal progress and fair choice. The maximal progress assumption allows the events/actions to occur when the components of the system are ready to perform them within a time period. We avoided the progress property violations by improving our operational semantics of enforcement mechanism with some missing domain preconditions and preconditions for actions. While checking the consistency of the policy, enforcement mechanism specifications, and the system properties, the policies with timing constraints can be enforced and the enforcement mechanism can be deployed in AEWRs; this can be used as a sound basis for the concrete implementation of the system, with high level of assurance and understanding.

Chapter 8: Conclusion

This chapter provides a summary of the work presented in this thesis and presents the research findings. We conclude this chapter with suggestions for future work.

8.1 Summary of the Thesis

Developing a framework for an adaptive early warning and response system to address insider privacy breaches that is compatible with a dynamic software system environment has been the overall aim of this thesis. This is needed due to the challenges that have been faced by the dynamic software system environment, where the insiders have direct access to information and are trusted by their organisations. However, the nature of insider privacy breaches fluctuates with the organisation's acceptable usage policy and the attributes of an insider. In a software system environment, such as e-government, the level of risk based on access patterns and timing constraints is considered in order to allow early warning against insider breaches. In addition to that, this aims to interrupt insider activity by enforcing adaptive response actions upon different scenarios of insider breaches. However, it is important to note that technical approaches alone are unable to detect and prevent insider breaches [22]. Hence, producing a framework for an adaptive early warning and response system to completely counter insider privacy breaches is challenging because insider behaviour may dynamically change based on intent and context. However, the necessity for a framework to support multiple and adaptive policy decisions at different risk levels in order to address the demands of a dynamic software system environment and to fill the gap of the existing frameworks has now been satisfied.

In meeting this goal, this thesis proposed a formal framework for an adaptive early warning and response system for insider privacy breaches. This framework allows for the specification of multiple policies at different risk levels during ongoing usage (depending on insider breach scenarios) and for enforcement of adaptive response

CHAPTER 8. CONCLUSION

actions to interrupt insider activity. Chapter 3 illustrated an architecture for an adaptive early warning and response system (AEWRS) for insider privacy breaches that extends the traditional UCON model to include interrupt policy decisions, where those decisions can be expressed at different risk levels. Interrupt policy decisions determine what interrupt response actions (skip, delay, suspend, or abort) will be executed depending on the level of early warning (very low, low, medium, or high), respectively, during access. In this architecture, an adaptive early warning and response component is integrated into the UCON model within a comprehensive approach in order to provide two layers of defence in depth that is insider-aware.

In addition, a computational model for AEWRS was presented in the form of Statechart to demonstrate the concurrent behaviour of the system, where concurrent data access requests can be handled. Building on policy-based management, we represented an abstraction of the enforcement mechanism model for an AEWRS system regulated by the policy rules of privacy breach specification language (PBSL) to determine the policy decision-making behaviour. The model gave a description of how the outcome of the policy rules, namely the early warning level, will affect the behaviour of the computational model by enabling the corresponding state transitions, namely interrupt response actions, changing the system states. Thus, it describes a link between the policy rules of PBSL and the system behaviour of AEWRS. The model demonstrated how policies can be adapted dynamically due to both the occurrence of events or actions and the passage of time. The development of AEWRS as an extension of UCON can be clearly seen from Statechart.

Chapter 4 presented PBSL based on the computational model. Within the language, privacy breach scenarios and the early warning level corresponding to the insider privacy breach and various timing constraints have been specified in a unified manner in the form of policy rules. The PBSL policy specifications are based on ECA rules. The standard ECA rules have been extended with various constructs, event-supporting patterns, and timing constraints. In our context, events are used to express patterns that contribute to a privacy breach scenario, while conditions are used to express the attribute constraints and timing constraints related to the breach scenario.

CHAPTER 8. CONCLUSION

Actions initiate the corresponding early warning level once a privacy breach scenario has been satisfied. The main features of PBSL are its expressiveness, simplicity, practicality, and formal semantics. The ability to express history-based behaviours using a high-level description in our specification language removed the necessity to use the notion of mutable attributes in UCON. Additionally, this increases the likelihood of the early detection of insider breaches with fewer false alarms. Furthermore, some examples of PBSL specifications with regard to some situations and examples of insider privacy breaches were provided to demonstrate the use of the language. These included data theft, masquerading, data inference, access for unauthorised purposes, and access without consent.

To add more rigour in the correctness of the specifications, LTS and asynchronous FLTL were considered suitable formalisms to specify operational semantics for PBSL and the enforcement mechanism model of AEWRs, which are time and event dependent and support the concurrency of execution. Additionally, asynchronous FLTL was deemed the most appropriate formalism to express the policies that are associated with timing constraints and to verify the (event-based) system properties on policy and enforcement mechanism (time event-based) models via the LTSA model checker.

The operational formal semantics of PBSL were described in LTS and FLTL (Chapter 5). The resulting LTS trace semantics of PBSL constructs were computed from asynchronous FLTL specifications using the LTSA animation facility. The graphical representations of PBSL constructs, including event patterns and timing constraints, were presented using timed LTS, which is based on state machines.

To facilitate communication between policy rules and the enforcement mechanism, we gave the formal operational semantics for the enforcement mechanism in LTS and asynchronous FLTL (Chapter 6). This was achieved as a means of defining enforcement rules and system behaviours, including their interactions, to allow for the enforcement of the PBSL policies. Given that AEWRs is adaptive, time-dependent, and

CHAPTER 8. CONCLUSION

security-critical within a concurrent setting, we proposed an enforcement mechanism design for the concurrent enforcement of policies using a synchronous interleaving model to ensure mutual exclusion and support clock and condition synchronisation.

Using a case study of a tax revenue system, we illustrated how the example scenario of insider breaches are identified and formalised from a given natural language. We demonstrated the benefits of our specification approach that utilises its expressiveness to specify complex policies. We demonstrated in Chapter 7 how the time progress property and safety properties of the system with respect to policy and enforcement mechanism specifications can be verified using the LTSA model checker. In checking the consistency of the policy, enforcement mechanism specifications and system properties, policies with timing constraints can be enforced, and the enforcement mechanism can be deployed in AEWRS. Thus, that can be used as a sound basis for the concrete implementation of the system, with a high level of assurance and understanding.

8.2 Contributions Revisited

The contributions of this thesis can be summarised as follows:

- 1- The UCON model has been extended to include interrupt policy decisions in which multiple policy decisions can be expressed at different risk levels during ongoing usage (**Chapter 3**).
- 2- A computational model has been developed to demonstrate the abstract concurrent behaviour of an adaptive early warning and response system for a real information system in the context of policy decision-making and enforcement (**Chapter 3**).
- 3- The Privacy Breach Specification language (PBSL) has been introduced (**Chapter 4**). This policy specification language has the following features:

CHAPTER 8. CONCLUSION

- **Expressiveness:** PBSL is able to express a wide range of complex events and timing constraints in a comprehensive scenario and can specify a wide range of policies within the system.
 - **Formal semantics:** PBSL is based on sound formal semantics. This allows the language's constructs and policy specifications to be analysed.
 - **Simplicity:** The constructs of PBSL are readable (easy to understand) and writable (easy to specify).
 - **Practicality:** The language can be implemented using an efficient mechanism.
- 4- The policy language and the mechanism enforcing the policies have been formalised in an operational style, allowing policies to be enforced on the system (**Chapter 5 and 6**).
- 5- The PBSL has been evaluated using a case study from the e-government domain (**Chapter 7**).
- 6- The safety and progress properties of the policy and enforcement mechanism models have been formally verified using an automatic verification tool (**Chapter 7**).

8.3 Future Work

Protection from Insider threats is a newly emerging and broached research area in the field of information security, and there are many issues that remain to be investigated. By providing a framework for an adaptive early warning and response system for insider privacy breaches, we have identified a number of suggestions to be considered in this regard. These are outlined below:

Complex Response Actions

As mentioned in Section 3.2, further complex response actions can be triggered within the delay or suspension period. These allow two or more response actions to run parallel or in sequence within the delay or suspend period. Some of these response actions are also suggested in [15], but no notation has been provided to express them. To define these complex response actions within PBSL, a response action language can be integrated into our language together with the temporal operators between actions as follows:

Construct	Description	Example
$A_1 ; A_2$	Sequence of response actions	<i>Block Data Transmission ; Abort</i>
$A_1 A_2$	Response actions in parallel	<i>delay Video Surveillance</i>
$A_1 \wedge A_2$	Response actions happen in any order	<i>suspend \wedge log</i>
$A_1 \rightarrow A_2$	Response actions happen and implies another action happens.	<i>delay \rightarrow alert</i>

Interrupt policy operator

To define interrupt policies within PBSL in a compositional fashion, interrupt policy constructs can be integrated into the language to express the interrupt policies using simple Boolean guards over state variables within different early warning levels. Interrupt policies can be expressed as $P \triangleright_t^e Q$, where a policy starts with the policy P . Then Q starts if either t time has elapsed or an event (interrupt) has occurred (that made the event e evaluate as true), whichever happens first. The abbreviations of the policy can be as follows, while the t and e are optional.

- $P \triangleright Q$

This is a policy that is equivalent to Q .

- $P \triangleright^e Q$

This is a policy that starts with P and only if an event e occurs, then Q takes over.

- $P \triangleright_t Q$

Here the interrupt policy starts with P and after a t time unit has elapsed, response action Q starts.

Examples.

$$\{ \text{Normal Operation} || \text{Video Surveillance} \triangleright_{60}^{\text{high EWL}} \text{Aborted} \}$$

$$\{ \text{Normal Operation} \triangleright^{\text{high EWL}} \text{Aborted} \}$$

Furthermore, we could introduce the expression $(P \triangleright_t^e Q) \trianglelefteq S$, where a policy starts with P . While P is operating, it waits for e event to occur for t time units. As soon as e occurs, Q policy starts. If after t time units and no e event has occurred or the event e evaluates to false, then S policy starts.

Example.

$$(\text{Suspended} \triangleright_{180}^{\text{Answerfromuser(Fingerprint,password)}} \text{Normal Operation}) \trianglelefteq \text{Aborted}$$

CHAPTER 8. CONCLUSION

Large Case Study

An area for future work is to use a large real-world case study with more complex scenario examples including insider breaches or fraud in collusion with each other using our PBSL language for the purposes of further evaluation.

Risk Engine

We can integrate a risk engine with our policy-based approach for dynamic risk evaluation to assess the risk level at the session and policy level. We can use a risk assessment matrix based on the likelihood of the breach scenario and potential impact. The level of risk can be calculated at run-time by multiplying the likelihood of each breach scenario by the potential impact from a particular insider to a specific information asset. The values of risk levels from different policy rules can be merged in order to determine the total value (e.g., average) of the risk level at the session level. A policy can be specified based on the final result in order to trigger the corresponding early warning action. The risk level can be calculated as follows:

the risk level from insider (I) against information asset (IA) = the likelihood of the breach scenario * damage impact.

The likelihood of an insider threat can be evaluated in order to identify the chance that a breach will happen to a digital information asset by an insider based on access patterns. The potential impact is the degree of potential damage or impact to an information asset in the case of a potential insider breach occurrence with regards to asset confidentiality and privacy.

Risk Level	Range
Very low early warning	$1 \leq \text{EWL} < 250$
Low early warning	$250 \leq \text{EWL} < 500$
Medium early warning	$500 \leq \text{EWL} < 750$
High early warning	$\text{EWL} > 750-1000$

Additional Extensions to UCON

Usage control remains a major open research area in information security. Hence, another issue of future work is to further investigate UCON to fully support insider threats with regards to specification and adopting efficient enforcement mechanisms and their implementation and deployment. These extensions include such as integrating the notions of context aware (e.g., location of the insider) in context-aware access control, and the purpose (the purpose of insider access) in purpose-aware access control.

Practicality

In order to demonstrate the applicability in practical settings, Implementation of enforcement mechanism to enforce PBSL policies tailored to specific applications and databases is an interesting area of future work. A related area is to investigate how the early warning mechanism that send signals to the enforcements mechanism can be implemented in a distributed setting with all the necessary facilities such as logging. To demonstrate the practicality in terms of the PBSL, another avenue of future work is to design a language and develop an easy-to use tool to enable end-users who interactively specify policies to use PBSL policies.

Integration with Information Flow Model

One important area of future work is to investigate how information flow model [16] can be adopted for our framework in order to control the flow of information after has been accessed from one insider to another. The system can then specify and enforce policies in order to detect illegal paths of information flow and prevent the information asset from being copied, printed or leaked outside the organisation.

Considering non-technical aspects associated with our approach

One important direction for future work is to accommodate nontechnical factors such as social, legal, ethical issues with our technical approach. These include AEWS needs to comply with data privacy laws such as the prohibition of use of monitoring personal data in AEWS for the purpose of monitoring person's job performance or analysing

CHAPTER 8. CONCLUSION

his/her personal habits [34], that is required to be used only for the purpose of breach early detection and warning.

References

- [1] M. Abadi and C. Fournet, "Access Control Based on Execution History", in *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS)*, 2003, p.107-121.
- [2] A. Act, "Health insurance portability and accountability act of 1996", *Public Law*, vol. 104, p. 191, 1996.
- [3] X. An, D. Jutla and N. Cercone, "Privacy intrusion detection using dynamic Bayesian networks", in *ICEC '06 Proceedings of the 8th international conference on Electronic commerce*. ACM, 2006.
- [4] R.H. Anderson, "Research and Development Initiatives Focused on Preventing, Detecting, and Responding to Insider Misuse of Critical Defense Information Systems: Results of a Three-Day Workshop", in *Proceedings of an August 1999 workshop*, RAND Corporation, 1999.
- [5] R.H. Anderson, T. Bozek, T. Longstaff, W. Meitzler and M. Skroch, "Research on mitigating the insider threat to information systems-# 2", in *Proceedings of an August 2000 workshop*, RAND Corporation, 2000.
- [6] B. Alpern and F.B. Schneider, "Defining liveness", *Information processing letters*, vol. 21, no. 4, p.181-185, 1985.
- [7] D. Alrajeh, J. Kramer, A. Russo and S. Uchitel, "Elaborating requirements using model checking and inductive learning", *IEEE Transactions on Software Engineering*, vol. 39, no. 3, p.361-383, 2013.

REFERENCES

- [8] D. Alrajeh, J. Kramer, A. Russo and S. Uchitel, "Deriving non-zeno behaviour models from goal models using ILP", *Formal aspects of computing*, vol. 22, no. 3-4, p.217-241, 2010.
- [9] P. Ashley, S. Hada, G. Karjoth, C. Powers and M. Schunter, *Enterprise privacy authorization language*, [EPAL 1.2]. Submission to W3C, 2003.
- [10] L. Audit Commission, *Ghost in the machine An analysis of IT fraud and abuse*. 1998, [Online] Available from: <http://archive.audit-commission.gov.uk/auditcommission/subwebs/publications/corporate/publicationPDF/1246.pdf> [Accessed 10/10/14].
- [11] A.K. Bandara, *A formal approach to analysis and refinement of policies*, PhD., Imperial College London (University of London), 2005.
- [12] A. Bandara, N. Damianou, E. Lupu, M. Sloman and N. Dulay, "Policy-based management", in J. Bergstra and M. Burgess, Eds., *Handbook of network and systems administration*. Elsevier, 2007, p.507-563.
- [13] D. Basin, V. Jugé, F. Klaedtke and E. Zălinescu, "Enforceable security policies revisited", *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 1, 2013.
- [14] BBC, "Six more data discs 'are missing' ", *BBC News*, 27th Nov 2007. [Online] Available from: <http://news.bbc.co.uk/1/hi/7111056.stm> [Accessed 15/10/2014].
- [15] E. Bertino, "Data Protection from Insider Threats", *Synthesis Lectures on Data Management*, vol. 4, no. 4, p.1-91, 2012.
- [16] M.A. Bishop, *The Art and Science of Computer Security*. Addison-Wesley Longman Publishing Co., Inc, 2002.

REFERENCES

- [17] M. Bishop, S. Engle, D.A. Frincke, C. Gates, F.L. Greitzer, S. Peisert and S. Whalen, "A risk management approach to the "insider threat", in C. W. Probst et al., Eds., *Insider Threats in Cyber Security*. Springer, 2010, p.115-137.
- [18] M. Bishop, D. Gollmann, J. Hunker and C. W. Probst, "Countering insider threats", in *Dagstuhl Seminar proceedings*, 2008, p.18.
- [19] R.C. Brackney and R.H. Anderson, "Understanding the Insider Threat", in *Proceedings of a March 2004 Workshop*, RAND Corporation, 2004.
- [20] D.F. Brewer and M.J. Nash, "The chinese wall security policy", in *1989 IEEE Symposium on Security and Privacy*, 1989, p.206-214.
- [21] S. Calo and J. Lobo, "A basis for comparing characteristics of policy systems", in *IEEE Workshop on Policies for Distributed Systems and Networks (POLICY 2006)*, IEEE, 2006.
- [22] D.M. Cappelli, A.P. Moore and R.F. Trzeciak, *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. Addison-Wesley, 2012.
- [23] Centre for the Protection of National Infrastructure (CPNI), *Insider Data Collection Study: Report of main Findings*. April 2013. [Online] Available from: <http://www.cpni.gov.uk/advice/Personnel-security1/Insider-threats/> [Accessed 10/01/15].
- [24] S. Chakravarthy, V. Krishnaprasad, E. Anwar and S. Kim, "Composite events for active databases: Semantics, contexts and detection", in *Proceedings of the 20th international conference on Very Large Data Bases (VLDB)*, 1994, p.606-617.

REFERENCES

- [25] J. Crampton and M. Huth, "Towards an access-control framework for countering insider threats", in C. W. Probst et al., Eds., *Insider Threats in Cyber Security*. Springer, 2010, p.173-195.
- [26] L. Cranor, M. Langheinrich and M. Marchiori, *A P3P preference exchange language 1.0*, [APPEL1. 0]. W3C working draft, 2002.
- [27] F. Cristian, "Probabilistic clock synchronization", *Distributed computing*, vol. 3, no. 3, p.146-158, 1989.
- [28] F. Cuppens and R. Ortalo, "LAMBDA: A language to model a database for detection of attacks", in *Recent advances in intrusion detection*, Springer, 2000.
- [29] N. Damianou, N. Dulay, E. Lupu and M. Sloman, "The ponder policy specification language", in M. Sloman et al., Eds., *Policies for Distributed Systems and Networks*. Springer, 2001, p.18-38.
- [30] A. Dardenne, A. Van Lamsweerde and S. Fickas, "Goal-directed requirements acquisition", *Science of computer programming*, vol. 20, no. 1, p.3-50, 1993.
- [31] R. Diaconescu and K. Futatsugi, *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. AMAST Series in Computing, World Scientific, 1998.
- [32] E. Directive, "95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data", *Official Journal of the EC*, vol. 23, no. 6, 1995.

REFERENCES

- [33] S.T. Eckmann, G. Vigna and R.A. Kemmerer, "STATL: An attack language for state-based intrusion detection", *Journal of computer security*, vol. 10, no. 1, p.71-103, 2002.
- [34] U. Flegel, F. Kerschbaum, P. Miseldine, G. Monakova, R. Wacker and F. Leymann, "Legally Sustainable Solutions for Privacy Issues in Collaborative Fraud Detection", in C. W. Probst et al., Eds., *Insider Threats in Cyber Security*. Springer, 2010, p.139-171.
- [35] W. Fokink, "Introduction to process algebra", 2nd ed. *Computer Science Monograph*. Springer-Verlag, 2007.
- [36] S. Gatzia and K.R. Dittrich, "SAMOS: An active object-oriented database system", *IEEE Data Eng. Bull.*, vol. 15, no. 1-4, p.23-26, 1992.
- [37] N.H. Gehani and H.V. Jagadish. "Ode as an Active Database: Constraints and Triggers", in *Proceedings of the 17th international conference on Very Large Data Bases (VLDB)*, 1991, p.327-336.
- [38] D. Giannakopoulou and J. Magee, "Fluent model checking for event-based systems", in *ACM SIGSOFT Software Engineering Notes*, ACM, 2003.
- [39] F.L. Greitzer, A. P. Moore, D. M. Cappelli, D. H. Andrews, L.A. Carroll and T.D. Hull, "Combating the insider cyber threat", *IEEE Security and Privacy*, vol. 6, no. 1, p.61-64, 2008.
- [40] J.V. Guttag, K. D. Jones, A. Modet and J.M. Wing, *Larch: languages and tools for formal specification*, New York: Springer-Verlag, 1993.
- [41] D. Harel, "Statecharts: A visual formalism for complex system", *Science of computer programming*, vol. 8, no. 3, p.231-274, 1987.

REFERENCES

- [42] 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", *IEEE Transactions on Software Engineering*, vol. 16, no. 4, p.403-414, 1990.
- [43] M. Hinchey, J.P. Bowen and C. A. Rouff, "Introduction to Formal Methods", in C. A. Rouffet et al., Eds., *Agent technology from a formal perspective*. Springer-Verlag, 2006, p.25-64.
- [44] C.A.R. Hoare, *Communicating sequential processes*. Prentice-hall Englewood Cliffs. 1985.
- [45] IAEA, *Preventive and Protective Measures against Insider Threats: Implementing guide*. Vienna: International Atomic Energy Agency, 2008.
- [46] H. Janicke, *The development of secure multi-agent systems*, PhD., 2007, De Montfort University.
- [47] H. Janicke, A. Cau, F. Siewe and H. Zedan, "Concurrent enforcement of usage control policies", in *IEEE Workshop on Policies for Distributed Systems and Networks (POLICY 2008)*, 2008, p.111-118.
- [48] H. Janicke, A. Cau, F. Siewe and H. Zedan, "Dynamic access control policies: specification and verification", *The Computer Journal*, p.bxs102, 2012.
- [49] H. Janicke, A. Cau and H. Zedan, "A note on the formalisation of UCON", in *Proceedings of the 12th ACM symposium on Access control models and technologies*, 2007.
- [50] C.B. Jones, *Systematic software development using VDM*. Prentice-Hall Englewood Cliffs, NJ, 1986.

REFERENCES

- [51] A. Kamra and E. Bertino, "Design and implementation of an intrusion response system for relational databases", *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 6, p.875-888, 2011.
- [52] B. Katt, M. Hafner and X. Zhang, "A usage control policy specification with petri nets", in *IEEE 5th International Conference on Collaborative Computing, Networking, Applications and Worksharing (CollaborateCom 2009)*, 2009, p.1-8.
- [53] E. Kowalski, T. Conway, S. Keverline, M. Williams, D. Cappelli, B. Willke and A. Moore, *Insider threat study: Illicit cyber activity in the government sector*. US Department of Homeland Security at the US Secret Service and CERT at the Software Engineering Institute (Carnegie Mellon University), 2008.
- [54] R. Koymans, "Specifying real-time properties with metric temporal logic", *Real-time systems*, vol. 2, no. 4, p.255-299, 1990.
- [55] S. Kumar, *Classification and detection of computer intrusions*, PhD., Purdue University, 1995.
- [56] L. Lamport, "Proving the correctness of multiprocess programs", *IEEE Transactions on Software Engineering*, vol. 3, no. 2, p.125-143, 1977.
- [57] L. Lamport, "The temporal logic of actions", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, p.872-923, 1994.
- [58] E. Letier, *Reasoning about agents in goal-oriented requirements engineering*, PhD., Université catholique de Louvain, 2001.

REFERENCES

- [59] E. Letier, J. Kramer, J. Magee and S. Uchitel, "Fluent temporal logic for discrete-time event-based models", in *ACM SIGSOFT Software Engineering Notes*, ACM, 2005.
- [60] E. Letier, J. Kramer, J. Magee and S. Uchitel, "Deriving event-based transition systems from goal-oriented requirements models", *Automated Software Engineering*, vol. 15, no. 2, p.175-206, 2008.
- [61] U. Lindqvist and P.A. Porras, "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)", in *Proceedings of the 1999 IEEE Symposium in Security and Privacy*, 1999.
- [62] J. Magee and J. Kramer, *Concurrency: state models & Java programs*, 2nd ed. John Wiley & Sons, 2006.
- [63] F. Martinelli, "A Model for Usage Control in GRID systems", in *Third IEEE International Conference on Security and Privacy in Communications Networks and the Workshops (SecureComm 2007)*, 2007, p.520-520.
- [64] J. Mattai and M. Joseph, *Real-Time Systems: specification, verification, and analysis*. Prentice Hall PTR, 1995.
- [65] M. McCormick, "Data theft: a prototypical insider threat", in J. S. Salvatore et al., Eds., *Insider Attack and Cyber Security: Beyond the Hacker*. Springer, 2008, p.53-68.
- [66] M. Meier, N. Bischof and T. Holz, "SHEDEL - A Simple Hierarchical Event Description Language for Specifying Attack Signatures", in *Proceedings of the 17th International Conference on Information Security*, Springer, 2002, p.559-571.

REFERENCES

- [67] C. Michel and L. Mé, "Adele: an attack description language for knowledge-based intrusion detection", in *Proceedings of the 16th International Conference on Information Security (IFIP/SEC 2001)*, Springer, 2001, p.353-368.
- [68] R. Milner, *Communication and concurrency*. Prentice-Hall, Inc, 1989.
- [69] R. Miller and M. Shanahan, "Some alternative formulations of the event calculus", in A. C. Kakas and F. Sadri, Eds., *Computational logic: logic programming and beyond*. Springer Berlin Heidelberg, 2002, p.452-490.
- [70] T. Moses, *Extensible access control markup language*, [xacml] version 2.0. Oasis Standard, 2005.
- [71] B. Moszkowski, "Executing temporal logic programs" in S. D. Brookes et al., Eds., *Seminar on concurrency*, Springer, 1985, p.111-130.
- [72] A. Mounji, *Languages and tools for rule-based distributed intrusion detection*, Belgium Doctor of Science Thesis, Facult es Universitaires Notre-Dame de la Paix, Namur, 1997.
- [73] P.G. Neumann, "Combatting insider threats", in C. W. Probst et al., Eds., *Insider Threats in Cyber Security*. Springer, 2010, p.17-44.
- [74] H.R. Nielson and F. Nielson, *Semantics with applications: a formal introduction*. John Wiley & Sons, Inc., 1992.
- [75] Office of Public Sector Information, *Data protection act 1998*. 1998.
- [76] J. Park and R. Sandhu, "The UCON ABC usage control model", *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, p.128-174, 2004.

REFERENCES

- [77] J. Park, X. Zhang and R. Sandhu, "Attribute mutability in usage control", in C. Farkas and P. Samarati, Eds., *Research Directions in Data and Applications Security XVIII*, Springer, 2004, p.15-29.
- [78] N.W. Paton and O. Díaz, "Active database systems", *ACM Computing Surveys (CSUR)*, vol. 31, no. 1, p.63-103, 1999.
- [79] J.L. Peterson, *Petri net theory and the modeling of systems*. Prentice-hall Englewood Cliffs (NJ), 1981.
- [80] T. Piper, " The Personal Information Protection and Electronic Documents Act-A Lost Opportunity to Democratize Canada's Technological Society", *Dalhousie LJ*, vol. 23, p. 253, 2000.
- [81] J.-P. Pouzol and M. Ducasé, "From declarative signatures to misuse IDS", in *Recent Advances in Intrusion Detection*, Springer, 2001.
- [82] C.W. Probst and R.R. Hansen, "An extensible analysable system model", *Information security technical report*, vol. 13, no. 4, p.235-246, 2008.
- [83] C.W. Probst, J. Hunker, D. Gollmann and M. Bishop, "Aspects of Insider Threats", in C. W. Probst et al., Eds., *Insider Threats in Cyber Security*. Springer, 2010, p.1-15.
- [84] D.E. Robling Denning, *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc, 1982.
- [85] M.B. Salem, S. Hershkop and S.J. Stolfo, "A survey of insider attack detection research", in J. S. Salvatore et al., Eds., *Insider Attack and Cyber Security: Beyond the Hacker*. Springer, 2008, p.69-90.

REFERENCES

- [86] R.S. Sandhu, E.J. Coyne, H. L. Feinstein and C.E. Youman, "Role-based access control models", *Computer*, vol. 29, no. 2, p.38-47, 1996.
- [87] F.B. Schneider, "Enforceable security policies", *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 1, p.30-50, 2000.
- [88] D. Schutzer, "Research Challenges for Fighting Insider Threat in the Financial Services Industry", in J. S. Salvatore et al., Eds., *Insider Attack and Cyber Security: Beyond the Hacker*. Springer, 2008, p.215-218.
- [89] F. Siewe, *A compositional framework for the development of secure access control systems*, PhD., De Montfort University, 2005.
- [90] M. Sloman, "Policy driven management for distributed systems", *Journal of network and Systems Management*, vol. 2, no. 4, p.333-360, 1994.
- [91] L. Spitzner, "Honeypots: Catching the insider threat", in *Proceedings of the IEEE Computer Security Applications Conference (19th Annual)*, 2003.
- [92] L. Spitzner, "Honeytokens: The other honeypot", 2003, [Online] Available from: http://bandwidthco.com/sf_whitepapers/honeypots/Honeytokens%20-%20The%20Other%20Honeypot.pdf [Accessed 13/10/14].
- [93] J.M. Spivey and J. Abrial, *The Z notation*. Prentice Hall Hemel Hempstead, 1992.
- [94] The Department for Business, Innovation and Skills (BIS), *Information Security Breaches Survey: Technical Report*. 2013, [Online] Available from: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/200455/bis-13-p184-2013-information-security-breaches-survey-technical-report.pdf [Accessed 13/01/15].

REFERENCES

- [95] The Department for Business, Innovation and Skills (BIS), *Information Security Breaches Survey: Technical Report*. 2014, [Online] Available from: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/307296/bis-14-767-information-security-breaches-survey-2014-technical-report-revision1.pdf [Accessed 13/01/15].
- [96] K. Twidle, E. Lupu, N. Dulay and M. Sloman, "Ponder2-a policy environment for autonomous pervasive systems", in *IEEE Workshop on Policies for Distributed Systems and Networks (POLICY 2008)*, IEEE, 2008, p.245-246.
- [97] H.S. Venter, M.S. Olivier and J.H. Eloff, "PIDS: a privacy intrusion detection system", *Internet Research*, vol. 14, no. 5, p.360-365, 2004.
- [98] Verizon, *2014 Data Breach Investigations Report*. 2014, [Online] Available from: <http://www.verizonenterprise.com/DBIR/2014/> [Accessed 16/01/15].
- [99] J. Yang, P. Ning, X. S. Wang and S. Jajodia, "CARDS: A distributed system for detecting coordinated attacks", in *Proceedings of the IFIP TC11 Fifteenth Annual Working Conference on Information Security for Global Information Infrastructures*, Springer, 2000, p.171-180.
- [100] X. Zhang, F. Parisi-Presicce and R. Sandhu, "Formal model and policy specification of usage control", *ACM Transactions on Information and System Security (TISSEC)*, vol. 8, no. 4, p.351-387, 2005.
- [101] X. Zhang, J. Park, F. Parisi-Presicce and R. Sandhu, "A logical specification for usage control", in *Proceedings of the ninth ACM symposium on Access control models and technologies*, 2004.

Glossary of Terms

Abort is an interrupt response action that disconnects the insider from the system, and immediately aborts the access process.

Abstraction Levels the abstraction level is related to the amount of detail represented in the model, and are supported by moving up and down the hierarchy of composed components of the model.

Acceptable Usage policy is a set of rules that limit the ways in which resources may be used and outlines the guidelines as to how they should be used.

Access Control is a process of limiting access to resources depending on the security policies, which determine whether to allow or deny subjects to access objects prior to access being executed.

Adaptive Early Warning and Response System AEWRS is proposed to mitigate and counter insider breaches by early detection; warn against insider breaches; delay, suspend, and interrupt the insider attacks. It consists of early warning policies and enforcement mechanisms.

Anomaly detection is intrusion detection technique that identifies the observed behaviour is abnormal when it deviates from the expected normal behaviour.

Asynchronous or Metric Temporal Logic - Asynchronous FLTL - expresses properties on the sequence of system states observed after the occurrence of an event by using temporal operators with bounded time intervals (qualitative notion of time).

Auditing is a process of logging and analysing access events to resources, to determine which incidents took place and who executed them.

Authentication is the process of verifying users' identities in a system, by determining who the user is or who he/she claims to be.

GLOSSARY

Authorisation is the process of determining the set of rights assigned to a user to execute access to particular objects.

Bounded Time operators are a form of operators that are restricted by time-bounds. It allow to specify and reason about properties by using a qualitative notion of time (bounded time interval).

Clock synchronisation refers to the technique that deals with the problem when concurrent processes signal events ordered based on the time, in order to enable those processes on different sub-systems to be aware of the passage of time and allow concurrent access to shared objects.

Condition synchronisation refers to the technique that allows the enforcement mechanism (monitor) to block processes until a particular condition holds.

Counterexamples are a sequence of execution traces produced by model checking which violate the property.

Delay is an interrupt response action that delays access for a period of time until timeout occurs.

Dense time model is a time model in which events occur at times that are represented by a dense continuous time domain such as real numbers.

Discrete time model is a time model in which events occur at times that are represented by a discrete time domain such as natural numbers.

Domain preconditions and Domain postcondition capture the elementary state transitions defined by the application of operations in that domain.

Early warning Action represents an operation that must be triggered once a privacy breach scenario has been detected.

Early warning levels are risk levels that are determined by the early warning level actions in the policy rules.

E-Government refers to the use of information technology by government agencies to improve the delivering of government services to citizens and to empower them to access to information.

GLOSSARY

Event patterns are complex events that occur when multiple primitive events occur according to some patterns.

False negative is the actual incident that goes undetected.

False positive or False Alarm is the incident that is detected as an actual attack although it is normal and not suspicious.

Fluents are time-varying properties of the world. A Fluent in FLTL can be either state-based or event-based predicate.

Fluent Linear Temporal Logic is a formalism based on a linear temporal logic to specify state-based and event-based properties on event-based models using the notion of “fluent”.

Formal Semantics is a description of the meaning of language syntax unambiguously in a formal mathematical way.

Formal verification is the process of applying a manual or automatic formal technique for establishing whether a given system satisfies a given property or behaves in accordance to some formal specification of the system.

Honeypot is designed to capture malicious users and trap attempts at inappropriate access of data.

Insider Fraud Insider use of IT for the unauthorized modification, addition, or deletion of an organization’s data (not programs or systems) for personal gain, or the theft of information that leads to an identity crime (identity theft, credit card fraud).

Insider IT Sabotage Insider use of information technology to direct harm at an organisation or an individual.

Insider Privacy Breach Insider use of IT for the access of personal data within an organisation in an unacceptable way for personal gain, or the theft of information that leads to an identity crime.

Insider Theft of Intellectual property (Espionage) Insider use of IT to steal proprietary information from the organisation, this category includes industrial espionage involving insiders.

GLOSSARY

Insiderness “Degree of Insiderness” the level of access a particular insider has with respect to a given asset, and measures the degree of access and knowledge of an asset, and the level of trust of the insider.

Interrupt policy decisions are proactive decisions at graded risk levels which can be dynamically adapted following an early warning to interrupt insider activity.

Interrupt Response action is an action that must be performed when an early warning level has been triggered, with the aim of responding to insider breaches in a timely manner.

Intrusion detection System is a system that identifies attempts to misuse or break into a computer system and reports them to the system administrator.

Labelled Transition System represents a system as a set of concurrent components, where each component represents a set of states and transitions between these states.

Labelled Transition System Analyser is a tool for modelling concurrency in form of state machines (LTS), and analysis using model checking with animation.

Liveness property asserts that something good eventually happens.

Misuse Detection is intrusion detection technique that identifies intrusions that conform to a predefined pattern of a known attack (misuse signature).

Model Checking is a widely used automated formal analysis technique for verifying the properties of finite-state concurrent systems.

Mutual exclusion refers to the requirement that no concurrent processes sharing object are in their critical section at the same time, so only one process must acquire a lock at a time.

Object is a passive entity that is typically associated with data structures in information systems such as records.

Policy specification is the process of expressing what response actions are triggered when some detected privacy breach scenario triggers a certain early warning level.

GLOSSARY

Policy Enforcement is the process of enforcing policies in the system by enforcement mechanisms. This mechanism causes a mutual interaction between the policy specifications and the system behaviour in order to ensure the consistency.

Privacy breach scenario is a known pattern or attack or vulnerability represented as a combination of events with some associated contextual information that, when detected, trigger the corresponding early warning level.

Privacy Breach Specification Language is a policy specification language that expresses privacy breach scenarios, and their the corresponding early warning levels as well as various timing constraints in a unified manner in the form of policy rules.

Progress property asserts that a particular action is often and eventually executed.

Safety property asserts that nothing bad happens.

Security Policy is a set of rules, laws and practices that regulate the use of resources and outline high-level guidelines as to allowing or denying access to sensitive information resources.

Subject is an active entity that performs actions on objects and is associated with users, or a processes acting on behalf of users.

Suspend is an interrupt response action that causes access to be suspended until some event occurs or the system times out.

Synchronous or standard temporal logic - synchronous FLTL - expresses properties on the sequence of system states observed at a fixed time rate by using a quantitative notion of time.

Required Precondition captures a permission to perform an action.

Required trigger condition captures an obligation to perform the action.

Risk Assessment is the process of risk identification, analysis, and evaluation.

Timing constraints are to constrain the period of time within which events may occur.

GLOSSARY

Usage Control is a usage model that encompasses traditional access control, trust management, and data rights management.

Video Surveillance is the use of video cameras to transmit a signal to a specific, limited set of monitors where it is used for detection of potential risks and crimes.