VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

LÊ HỒNG ANH

METHODS FOR MODELING AND VERIFYING EVENT-DRIVEN SYSTEMS

DOTORAL THESIS IN INFORMATION TECHNOLOGY

Hà Nội – 2015

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Lê Hồng Anh

METHODS FOR MODELING AND VERIFYING EVENT-DRIVEN SYSTEMS

Major: Software Engineering
Mã số: 62.48.01.03

DOCTORAL THESIS IN INFORMATION TECHNOLOGY

SUPERVISORS:
1. Assoc. Prof. Dr. Trương Ninh Thuận
2. Assoc. Prof. Dr. Phạm Bảo Sơn

Hà Nội – 2015

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Lê Hồng Anh

PHƯƠNG PHÁP MÔ HÌNH HÓA VÀ KIỂM CHỨNG CÁC HỆ THỐNG HƯỚNG SỰ KIỆN

Chuyên ngành: Kỹ thuật phần mềm
Mã số: 62.48.01.03

LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:
1. PGS. TS. Trương Ninh Thuận
2. PGS. TS. Phạm Bảo Sơn

Hà Nội – 2015

Declaration of Authorship

I declare that this thesis titled, 'Methods for modeling and verifying event-driven systems' and the work presented in it are my own. I confirm that:

- I have acknowledged all main sources of help. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.
- This work was done wholly while in studying for a PhD degree

Signed:

Date:

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Lê Hồng Anh

METHODS FOR MODELING AND VERIFYING EVENT-DRIVEN SYSTEMS

Major: Software Engineering
Mã số: 62.48.01.03

DOCTORAL THESIS IN INFORMATION TECHNOLOGY

SUPERVISORS:
1. Assoc. Prof. Dr. Trương Ninh Thuận
2. Assoc. Prof. Dr. Phạm Bảo Sơn

Hà Nội – 2015

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Lê Hồng Anh

PHƯƠNG PHÁP MÔ HÌNH HÓA VÀ KIỂM CHỨNG CÁC HỆ THỐNG HƯỚNG SỰ KIỆN

Chuyên ngành: Kỹ thuật phần mềm
Mã số: 62.48.01.03

LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:
1. PGS. TS. Trương Ninh Thuận
2. PGS. TS. Phạm Bảo Sơn

Hà Nội – 2015

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Lê Hồng Anh

METHODS FOR MODELING AND VERIFYING EVENT-DRIVEN SYSTEMS

Major: Software Engineering
Mã số: 62.48.01.03

DOCTORAL THESIS IN INFORMATION TECHNOLOGY

SUPERVISORS:
1. Assoc. Prof. Dr. Trương Ninh Thuận
2. Assoc. Prof. Dr. Phạm Bảo Sơn

Hà Nội – 2015

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Lê Hồng Anh

PHƯƠNG PHÁP MÔ HÌNH HÓA VÀ KIỂM CHỨNG CÁC HỆ THỐNG HƯỚNG SỰ KIỆN

Chuyên ngành: Kỹ thuật phần mềm
Mã số: 62.48.01.03

LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:
1. PGS. TS. Trương Ninh Thuận
2. PGS. TS. Phạm Bảo Sơn

Hà Nội – 2015

Abstract

Modeling and verification plays an important role in software engineering because it improves the reliability of software systems. Software development technologies introduce a variety of methods or architectural styles. Each system based on a different architecture is often proposed with different suitable approaches to verify its correctness. Among these architectures, the field of event-driven architecture is broad in both academia and industry resulting the amount of work on modeling and verification of event-driven systems.

The goals of this thesis are to propose effective methods for modeling and verification of event-driven systems that react to emitted events using Event-Condition-Action (ECA) rules and Fuzzy If-Then rules. This thesis considers the particular characteristics and the special issues attaching with specific types such as database and context-aware systems, then uses Event-B and its supporting tools to analyze these systems.

First, we introduce a new method to formalize a database system including triggers by proposing a set of rules for translating database elements to Event-B constructs. After the modeling, we can formally check the data constraint preservation property and detect the infinite loops of the system.

Second, the thesis proposes a method which employs Event-B refinement for incrementally modeling and verifying context-aware systems which also use ECA rules to adapt the context situation changes. Context constraints preservation are proved automatically with the Rodin tool.

Third, the thesis works further on modeling event-driven systems whose behavior is specified by Fuzzy If-Then rules. We present a refinement-based approach to modeling both discrete and timed systems described with imprecise requirements.

Finally, we make use of Event-B refinement and existing reasoning methods to verify both safety and eventuality properties of imprecise systems requirements.

Acknowledgements

First of all, I would like to express my sincere gratitude to my first supervisor Assoc. Prof. Dr. Truong Ninh Thuan and my second supervisor Assoc. Prof. Pham Bao Son for their support and guidance. They not only teach me how to conduct research work but also show me how to find passion on science.

Besides my supervisors, I also would like to thank Assoc. Prof. Dr. Nguyen Viet Ha and lecturers at Software Engineering department for their valuable comments about my research work in each seminar.

I would like to thank Professor Shin Nakajima for his support and guidance during my internship research at National Institute of Informatics, Japan.

My sincere thanks also goes to Hanoi University of Mining and Geology and my colleges there for their support during my PhD study.

Last but not least, I would like to thank my family: my parents, my wife, my children for their unconditional support in every aspect. I would not complete the thesis without their encouragement.

. . .

Contents

Declaration of Authorship **i**

Abstract **ii**

Acknowledgements **iii**

Table of Contents **iv**

List of Abbreviations **viii**

List of Tables **ix**

List of Figures **x**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 6
- 1.3 Literature review . . . 7
- 1.4 Contributions . . . 10
- 1.5 Thesis structure . . . 11

2 Backgrounds **13**
- 2.1 Temporal logic . . . 13
- 2.2 Classical set theory . . . 15
- 2.3 Fuzzy sets and Fuzzy If-Then rules . . . 17
  - 2.3.1 Fuzzy sets . . . 17
  - 2.3.2 Fuzzy If-Then rules . . . 18
- 2.4 Formal methods . . . 19
  - 2.4.1 VDM . . . 21
  - 2.4.2 Z . . . 23
  - 2.4.3 B method . . . 24
- 2.5 Event-B . . . 27
  - 2.5.1 An overview . . . 27

2.5.2 Event-B context 28
2.5.3 Event-B Machine 29
2.5.4 Event-B mathematical language 31
2.5.5 Refinement 32
2.5.6 Proof obligations 33
2.6 Rodin tool 36
2.7 Event-driven systems 37
2.7.1 Event-driven architecture 37
2.7.2 Database systems and database triggers 38
2.7.3 Context-aware systems 40
2.8 Chapter conclusions 42

3 Modeling and verifying database trigger systems 44
3.1 Introduction 44
3.2 Related work 47
3.3 Modeling and verifying database triggers system 48
3.3.1 Modeling database systems 49
3.3.2 Formalizing triggers 50
3.3.3 Verifying system properties 53
3.4 A case study: Human resources management application 54
3.4.1 Scenario description 54
3.4.2 Scenario modeling 55
3.4.3 Checking properties 57
3.5 Support tool: Trigger2B 59
3.5.1 Architecture 59
3.5.2 Implementation 60
3.6 Chapter conclusions 62

4 Modeling and verifying context-aware systems 64
4.1 Introduction 64
4.2 Related work 66
4.3 Formalizing context awareness 67
4.3.1 Set representation of context awareness 68
4.3.2 Modeling context-aware system 69
4.3.3 Incremental modeling using refinement 71
4.4 A case study: Adaptive Cruise Control system 72
4.4.1 Initial description 73
4.4.2 Modeling ACC system 73
4.4.3 Refinement: Adding weather and road sensors 75
4.4.4 Verifying the system's properties 78
4.5 Chapter conclusions 78

5 Modeling and verifying imprecise system requirements 81
5.1 Introduction 81
5.2 Related work 83

5.3 Modeling fuzzy requirements . . . 85
5.3.1 Representation of fuzzy terms in classical sets . . . 85
5.3.2 Modeling discrete states . . . 87
5.3.3 Modeling continuous behavior . . . 88
5.4 Verifying safety and eventuality properties . . . 91
5.4.1 Convergence in Event-B . . . 91
5.4.2 Safety and eventuality analysis in Event-B . . . 92
5.4.3 Verifying safety properties . . . 93
5.4.4 Verifying eventuality properties . . . 94
5.5 A case study: Container Crane Control . . . 98
5.5.1 Scenario description . . . 98
5.5.2 Modeling the Crane Container Control system . . . 100
5.5.2.1 Modeling discrete behavior . . . 100
5.5.2.2 First Refinement: Modeling continuous behavior . . . 102
5.5.2.3 Second Refinement: Modeling eventuality property . . . 104
5.5.3 Checking properties . . . 106
5.6 Chapter conclusions . . . 108

6 Conclusions **109**
6.1 Achievements . . . 109
6.2 Limitations . . . 113
6.3 Future work . . . 114

List of Publications **116**

Bibliography **117**

A Event-B specification of Trigger example **128**
A.1 Context specification of Trigger example . . . 128
A.2 Machine specification of Trigger example . . . 129

B Event-B specification of the ACC system **132**
B.1 Context specification of ACC system . . . 132
B.2 Machine specification of ACC system . . . 133
B.3 Extended context . . . 134
B.4 Refined machine . . . 134

C Event-B specifications and proof obligations of Crane Controller Example **136**
C.1 Context specification of Crane Controller system . . . 136
C.2 Extended context . . . 137
C.3 Machine specification of Crane Controller system . . . 138
C.4 Refined machine . . . 140
C.5 Proof obligations for checking the safety property . . . 143

C.6 Proof obligations for checking convergence properties 144

List of Abbreviations

DDL	**D**ata **D**afinition **L**anguage
DML	**D**ata **M**anipulation **L**anguage
PO	**P**roof **O**bligation
LTL	**L**inear **T**emporal **L**ogic
SCR	**S**oftware **C**ost **R**eduction
ECA	**E**vent **C**ondition **A**ction
VDM	**V**ienna **D**evelopment **M**ethod
VDM-SL	**V**ienna **D**evelopment **M**ethod - **S**pecification **L**anguage
FM	**F**ormal **M**ethod
PTL	**P**ropositional **T**emporal **L**ogic
CTL	**C**omputational **T**emporal **L**ogic
SCR	**S**oftware **C**ost **R**eduction
AMN	**A**bstract **M**achine **N**otation

List of Tables

2.1 Truth tables for propositional operators 14
2.2 Meaning of temporal operators 15
2.3 Truth table of implication operator 19
2.4 Comparison of B, Z and VDM [1] 27
2.5 Relations and functions in Event-B 32
2.6 INV proof obligation 34
2.7 VAR PO with numeric variant 35
2.8 VAR PO with finite set variant 35

3.1 Translation rules between database and Event-B 50
3.2 Formalizing a trigger 51
3.3 Encoding trigger actions 53
3.4 Table EMPLOYEES and BONUS 55
3.5 INV PO of event $trigger1$. 58
3.6 Infinite loop proof obligation of event $trigger1$ 59

4.1 Modeling a context rule by an Event-B Event 70
4.2 Transformation between context-aware systems and Event-B 70
4.3 Proof of context constraint preservation 78

5.1 INV PO of event $evt4$ 106
5.2 Deadlock free PO of machine $Crane\_M1$ 107
5.3 VAR PO of event $evt4$ 108

C.1 INV PO of event $evt1$ 143
C.2 INV PO of event $evt2$ 143
C.3 INV PO of event $evt3$ 143
C.4 INV PO of event $evt5$ 144
C.5 VAR PO of event $evt1$ 144
C.6 NAT PO of event $evt1$ 144
C.7 VAR PO of event $evt2$ 144
C.8 NAT PO of event $evt2$ 145
C.9 VAR PO of event $evt3$ 145
C.10 NAT PO of event $evt3$ 145
C.11 VAR PO of event $evt5$ 145
C.12 NAT PO of event $evt5$ 145

List of Figures

1.1 Types of event-driven systems 3
1.2 Thesis structure 12

2.1 Basic structure of an Event B model 28
2.2 An Event-B context example 29
2.3 Forms of Event-B Events 30
2.4 Event-B refinement 32
2.5 Event refinement in Event-B 33
2.6 A convergent event 35
2.7 The Rodin tool 36
2.8 A layered conceptual framework for context-aware systems [2] 41

3.1 Partial Event-B specification for a database system 51
3.2 A part of Event-B Context 56
3.3 A part of Event-B machine 57
3.4 Encoding trigger 58
3.5 Architecture of Trigger2B tool 60
3.6 A partial parsed tree syntax of a general trigger 61
3.7 The modeling result of the scenario generated by Trigger2B 62

4.1 A simple context-aware system 68
4.2 Incremental modeling using refinement 71
4.3 Abstract Event-B model for ACC system 75
4.4 Events with strengthened guards 76
4.5 Refined Event-B model for ACC system 77
4.6 Checking properties in Rodin 79

5.1 A part of Event-B specification for discrete transitions modeling 89
5.2 A part of Event-B specification for continuous transitions modeling . . 90
5.3 A part of Event-B specification for eventuality property modeling . . . 96
5.4 Container Crane Control system 98
5.5 Safety properties are ensured in the Rodin tool automatically 107

Chapter 1

Introduction

1.1 Motivation

Nowadays, software systems become more complex and can be used to integrate with other systems. Software engineers need to understand as much as possible what they are developing. Modeling is one of effective ways to handle the complexity of software development that allows to design and assess the system requirements. Modeling not only represents the content visually but also provides textual content. There are several types of modeling language including graphical, textual, algebraic languages.

In software systems, errors may cause many damages for not only economics but also human beings, especially those applications in embedded systems, transportation control and health service equipment, etc. The error usually occurs when the system execution cannot satisfy the characteristics and constraints of the software system specification. The specification is the description of the required functionality and behavior of the software. Therefore, ensuring the correctness of software systems has always been a challenge of software development process and reliability plays an important role deciding the success of a software project.

Testing techniques are used in normal development in order to check whether the software execution satisfies users requirements. However, testing is an incomplete validation because it can only identifies errors but can not ensure that the software execution is correct in all cases.

Software verification is one of powerful methods to find or mathematically prove the absent of software errors. Several techniques and methods have been proposed for software verification such as model-checking [3], theorem-proving [4] and program analysis [5]. Among these techniques, theorem proving has distinct advantages such as superior size of the system and its ability to reason inductively. Though, theorem proving often generates a lot of proofs which are complex to understand. Verification techniques mainly can be classified into two kinds: model-level and implementation level. Early verification of model specifications helps to reduce the cost of software construction. For this reason, modeling and verification of software systems are an emerging research topic in around the world. Many approaches and techniques of modeling and verification have been proposed so far. Each of them usually focuses on a typical kind of software architecture or design styles.

In a traditional system, one component provides a collection of procedures and functions via its interfaces. Components then interact with each other by explicitly invoking those routines. Event-driven architecture is one of the most popular architectures in software project development providing implicit invocation instead of invoking routines directly. Each component of an event-driven system can produce events, the system then invoke all procedures which are registered with these events. An event-driven system consists of three essential parts: monitoring component, transmission component and responsive one. Since such systems work by raising and responding to events, it looses coupling between software components and improves the interactive capabilities with its environment. The event-driven architectural style is becoming an essential part of large-scale distributed systems design and many applications. It is a promising architecture to develop and model loosely coupled systems and its advantages have been recognized in both academia and industry.

There are many types of event-driven systems including many editors where user interface events signify editing commands, rule-based production systems where a condition becoming true causes an action to be triggered and active objects where changing a value of an object's attribute triggers some actions (e.g. database trigger systems) [6]. Figure 1.1 shows the hierarchy of listed event-driven systems. In this thesis, we consider two applications of active objects and rule-based production systems: database systems with triggers and context-aware systems.

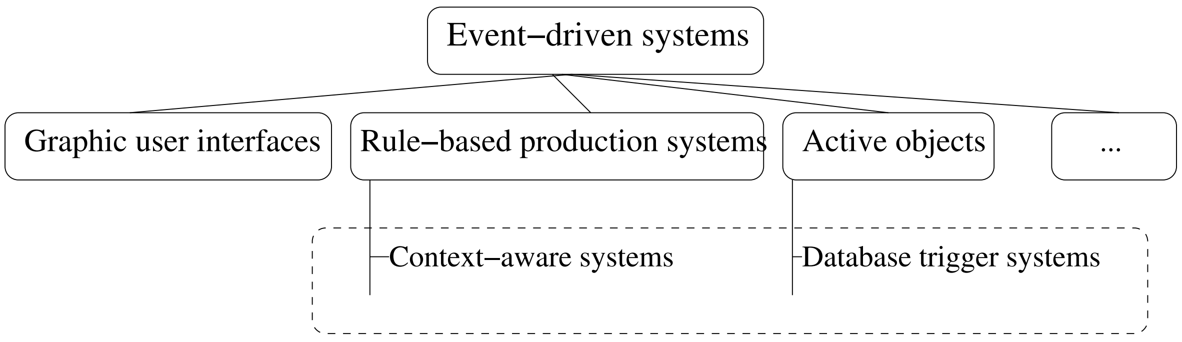


FIGURE 1.1: Types of event-driven systems

In event-driven systems, Event-Condition-Action (ECA) rules are proposed as a declarative approach to specify relations when certain events occur at predefined conditions. An ECA rule has the form: On *Event*

IF *conditions* DO *actions* that means when Events occurs, if *conditions* holds, then *actions* is performed. We also can informally represent it by if-then rules such as **if** *Events* occurs and *condition* holds, **then** perform *action*. The advantages of this approach have been applied and incorporated in various application domains such as active database systems, context-aware applications. There are a huge amount of studies working on analysing event-driven systems as well as formalizing ECA rules.

Researchers have proposed many approaches to modeling and verifying both centralized and distributed event-driven systems with model checking techniques, which are based on temporal logic and computational logic. Madl [7] presented an approach that defines a specification of a formal semantic domain and proposed a model-checking method to verify distributed real-time embedded systems based on timed-automata. Joanne Atlee and John Gannon [8] focused on formalizing event-driven system requirements based on computational tree logic (CTL). I. Ray and P.Annmann [9] proposed to use the B-Toolkit to detect safety violations in an example of software cost reduction (SCR) specification. Fiege *et al.* [10] presented a formal specification of scopes and event mappings within a trace-based formalism adapted from temporal logic. Tran and Zdun [11] introduced formal specification of the event actors-based constructs and the graphical notations based on Petri nets in order to enable formal analysis of such constructs. Calder and Savegnani [12] employed a universal process algebra that encapsulates both dynamic and spatial behaviour to extend and introduce a basic formalism of bi-graphical reactive systems.

These approaches have been proposed to modeling and verifying general even-driven systems. In fact, engineers often develop particular types of event-driven systems which use ECA rules to react to raised events, e.g., active databases and context-aware systems. In this case, these general approaches are insufficient. Furthermore, almost existing work of software verification focuses on analysing precise descriptions of the system's functionalities and behavior. There are a few of methods for verifying event-driven systems which are described by vague, uncertain or imprecise requirements.

For these reasons, new suitable methods for modeling and verifying such systems are desirable. Moreover, if we can verify significant properties of the system at early stage of design time, it will reduce cost of the system development. It is also beneficial if they reduce the complexity of proving and is practical in software development. The thesis proposes new methods to achieve that desire by using Event-B formal method [13]. It is an evolution of the B formalism [14] which was developed more than ten years ago and which has been applied in the number of industrial projects. Many researchers and research groups around the world have been inspired by system modeling and verification with Event-B. Hundreds of publications relating to Event-B have been published since 2004 [15]. Event-B notations are based on set theory, generalized substitutions and the first order logic. It is more suitable for developing large reactive and distributed systems. Software development in Event-B begins by abstractly specifying the requirements of the whole system, then refines them through several steps to reach a description of the system in such a detail that can be translated into code. The consistency of each model and the relationship between an abstract model and its refinements are obtained by formal proofs. Support tools have been provided for Event-B specification and proof in the Rodin platform [16]. Hence, Event-B is totally matched for modeling and verifying event-driven systems.

1.2 Objectives

The thesis aims to provide new and effective approaches in comparison with the existing work. Instead of working on analysing a general event-driven system or proposing any new formal language of ECA, we focus on modeling and verifying specific domain applications of the event-driven architecture such as database systems and context-aware systems using Event-B. The thesis objective is proposing methods that not only model the behavior of these systems which are described by If-Then rules (ECA rules) but also formalize significant properties by Event-B constructs. The correctness of these properties are proved mathematically by proving the Event-B generated proof obligations. The Rodin tool is used for supporting modeling and verification process to reduce the complexity with automatic proving.

The thesis directs at providing tools, which support for automatic translation from an application of event-driven systems to a target Event-B model that makes less effort and reduces the difficulties in modeling process. The output of these tools are expected to be able usable in the Event-B supporting tools such as Rodin.

The thesis has another objective to analyse event-driven systems whose behavior is described by imprecise requirements. These requirements are represented by Fuzzy If-Then rules. The thesis introduces a new refinement-based method for modeling imprecise requirements and verifying the significant properties such as safety and eventuality properties of such systems.

1.3 Literature review

Joanne Atlee and John Gannon [8] presented an approach to checking event driven systems using model checking. They introduced a technique to transforming event-oriented system requirements into state-based structures, then used a state-based checker to analyse. This method can detect violations of systems invariants. However, it is general approach therefore if we want to apply in a specific domain the one is not easy to follow. Moreover, it is inefficient when requiring intermediate steps to translate requirement into CTL machines.

Similarly, Ray I. and Ammann P. [9] also checked safety properties of event-driven systems using B-Toolkit [17]. Even though this method can translates SRC requirements to an Abstract machine notations (AMN) machine [14] directly, it is still too abstract to apply in a specific domain and has several limitations such as requiring developers to understand SCR and target B model contains only single class.

Prashanth, C.M. [18] described an efficient method to detect safety specification violations in dynamic behavior model of concurrent/reactive systems. The dynamic behavior of each concurrent object in a reactive system is assumed to be represented using UML (Unified Modeling Language) state chart diagram. The verification process involves building a global state space graph from these independent state chart diagrams and traversal of large number of states in global state space graph for detecting a safety violation.

Jalili, S. and Mirzaaghaei, M. [19] proposed to use event-based real-time logic (ERL) as a specification language in order to simply specify safety properties. By applying aspect-oriented approach to instrumentation, we can integrate runtime verification module (i.e. Monitor) with program itself and minimize overhead of runtime verification too. The method, called RVERL, consists of three phases. First, safety properties are extracted from program requirements specification. Second, properties are mapped to timing, functional and deadline aspects which constitute the monitor. Then it is weaved to the program source code. Third, at the execution time, the monitor observes program behavior and prevent it from property violations.

Amorim Marcelo and Havelund Klaus [20] introduced the temporal logic HAWK, a programming-oriented extension of the rule-based EAGLE logic, and its supporting tool for runtime verification of Java programs. A monitor for a HAWK formula checks if a finite trace of program events satisfies the formula. It has been shown capable of defining and implementing a range of finite trace monitoring logics, including future and past time temporal logic, metric (real-time) temporal logics, interval logics, forms of quantified temporal logics, extended regular expressions, state machines, and others. Monitoring is achieved on a state-by-state basis avoiding any need to store the input trace. HAWK extends EAGLE with constructs for capturing parameterized program events such as method calls and method returns.

Tran and Zeduh [11] introduced formal specification of the event actors-based constructs and the graphical notations based on Petri nets in order to enable formal analysis of such constructs. Based on this, an automated translation from event actors based constructs to Petri nets using template-based model transformation techniques is also developed.

Feideiro *et al.* [21] proposed a mathematical semantics for event-based architectures to characterize the modularization properties and to further validate and extend the categorical approach to architectural modeling.

Posse E. *et al.* [22, 23] proposed a language for modeling, analysis and simulation of time-sensitive, event-driven systems. It is a language from an informal perspective and discuss its implementation based on event-scheduling and time-warp for distributed simulation.

Calder M. *et al.* [12] employ a universal process algebra that encapsulates both dynamic and spatial behaviour to extend and introduce a basic formalism of bi-graphical reactive systems. They presented a case study involving wireless home network management and the automatic generation of bi-graphical models, and their analysis in real-time.

Baouab Aymen *et al.* [24] proposed new components, to be deployed along the boundaries of each participating organization, offering external flow control, and notification in case of violation detection, while providing process execution traceability. Then they proposed an event-based approach in which inter-organizational exchanges are perceived as events and define event patterns for filtering the desirable incoming and outgoing messages.

These approaches and methods can be classified into two categories: model-level verification and implementation-level verification. In this thesis, we focus on the latter because it helps to detect errors in early design phase. With the listed model-level verification methods, the common problems are that they are too general to apply in specific domains.

The research results relating to modeling and verifying the specific types event-driven systems are discussed in detail in Chapter 3, Chapter 4, and Chapter 5.

1.4 Contributions

Research contributions of this thesis are as follows.

1. This thesis introduces a new method to model and verify a database trigger system using Event-B. This approach provides detailed steps to translate database concepts to Event-B notations. The translation is based on the similarity between triggers which has the form of ECA rules and Event-B events. The method reduces cost of development because it can detect errors at early design phase and it is easy to apply in practice. A tool partly supports for transforming a database system with triggers is also developed.

2. The thesis continues investigating the benefit of similar acts between ECA rules and Event-B events to propose a method to model and verify context-aware systems. Furthermore, the thesis recognizes the advantages of Event-B refinement mechanism to make the proposed method suitable for incremental modeling. Significant properties, e.g., context constraints, are defined as invariants and can be checked automatically using the supporting tool Rodin.

3. We consider a system, which is described by imprecise requirements. Its behavior rules are now specified in the form of Fuzzy If-Then rules. The thesis introduces a new representation of fuzzy terms by classical sets and present a set of rules to translate Fuzzy If-Then rules to Event-B constructs. We also make an extension by introducing timed Fuzzy If-Then rules to model a timed system. The thesis makes use of Event-B refinement and the proposed modeling method to analyse some significant properties of imprecise system requirements such as safety and eventuality properties.

1.5 Thesis structure

The remainder of this thesis is organized as follows.

Chapter 2 provides necessary backgrounds for the thesis. Firstly, we briefly introduce about temporal logic, fuzzy sets and fuzzy If-Then rules. Next, an overview of formal verification and some formal methods such as VDM, Z and B is introduced. Event-B method and its supporting tool RODIN then is discussed in more detail. An overview of event-driven systems and their applications such as relational database systems and context-aware systems are also given.

Next, Chapter 3 shows how a database system including triggers can be modeled and verified using Event-B. We propose a new method with a set of translation rules to translate database components to Event-B notations. The method ensures the correctness of data constraint preservation and enables us to discover infinite loops of the trigger execution. A tool which partly supports for the modeling process is also developed.

We focus on modeling and verifying context-aware systems which use context rules reacts to context changes in Chapter 4. A set of translation rules for mapping context-aware components and Event-B are presented. It is a refinement-based method that allows to model the system gradually. After modeling, safety properties of these systems are proved formally.

In Chapter 5, we consider the case that an event-driven system is described with imprecise requirements, i.e., its behavior can be described by Fuzzy If-Then rules. We propose a new representation of fuzzy terms in classical sets and a refinement-based method to model both discrete and continuous behavior of the system. We also present a new method which makes use of Event-B refinement and existing methods to verify safety and eventuality properties of imprecise system requirements. We show that the verification is mostly conducted automatically using the current RODIN tool.

Finally, Chapter 6 discusses contributions and limitations of the thesis. It concludes and outlines the future research direction of the thesis. The organization of the thesis is illustrated in Figure 1.2.

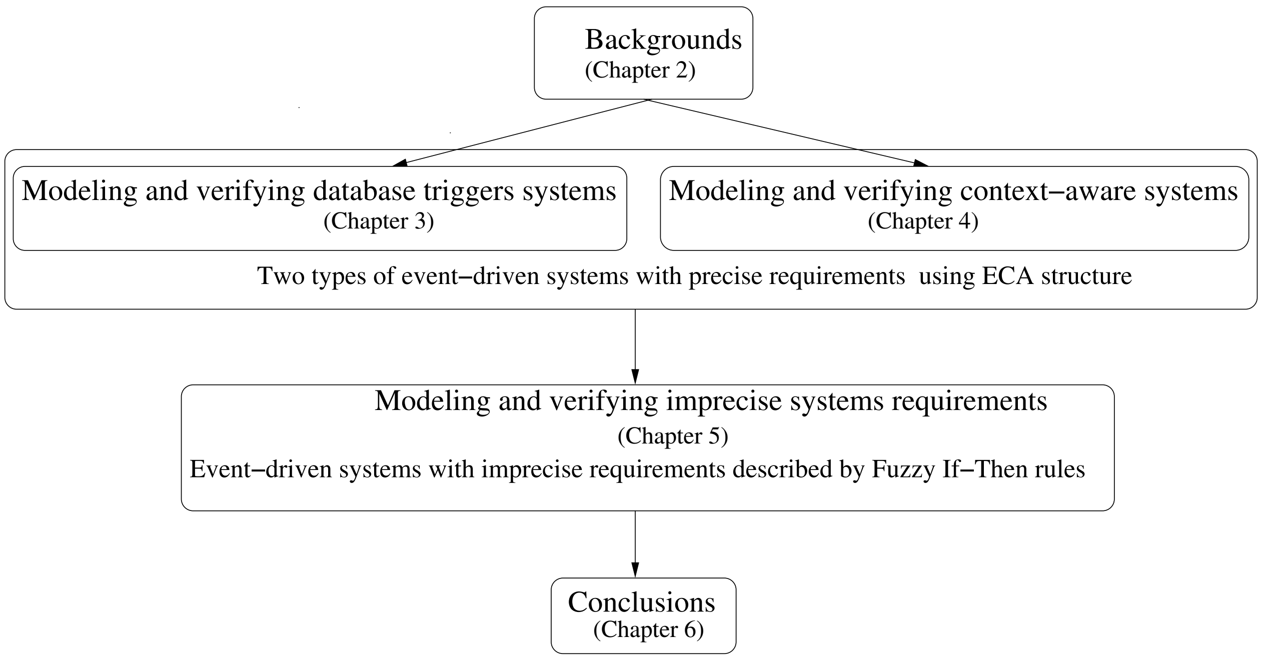

Backgrounds
(Chapter 2)
Modeling and verifying database triggers systems
(Chapter 3)
Modeling and verifying context–aware systems
(Chapter 4)
Two types of event–driven systems with precise requirements using ECA structure
Modeling and verifying imprecise systems requirements
(Chapter 5)
Event–driven systems with imprecise requirements described by Fuzzy If–Then rules
Conclusions
(Chapter 6)

FIGURE 1.2: Thesis structure

Chapter 2

Backgrounds

In this chapter, we provide the related background knowledge for the thesis. We first give a brief introduction of mathematical knowledge and logic such as classical set theory, fuzzy sets and temporal logic. After that, before presenting Event-B formal method in detail, an overview of VDM and its two ancestor formal methods such as Z and B is also given. Finally, we introduce the background of event-driven architecture and its application in two domains such as database and context-aware computing.

2.1 Temporal logic

In classical propositional logic, a proposition is evaluated to either *true* ($\top$) or *false* ($\perp$) A propositional formula is a syntactic expression built from a set of atomic predicates also sometimes known as atomic propositions, boolean variables, or simply predicates which we denote by lower case letters: p, q, etc. The most simple propositional formula is an expression known as an atom which merely consists of a single atomic predicate.

More complex formulas are then built from atoms using the connectives such as *conjunction* ($\wedge$) , *disjunction* ($\vee$), *negation* ($\neg$), *implication* ($\Rightarrow$), *equivalence* ($\Leftrightarrow$). Table 2.1 defines the meaning of the propositional logic operators.

TABLE 2.1: Truth tables for propositional operators

$\wedge$	$\bot$	$\top$
$\bot$	$\bot$	$\bot$
$\top$	$\bot$	$\top$

$\vee$	$\bot$	$\top$
$\bot$	$\bot$	$\top$
$\top$	$\top$	$\top$

	$\neg$
$\bot$	$\top$
$\top$	$\bot$

A serious disadvantage of propositional logic is that it cannot be used to describe the time-dependent behavior of the system [25]. Propositional temporal logic (PTL) extends the descriptive power of propositional logic to describe a sequence of states in different moments of time called time instants. We assume that there is one designated time instant representing the present. Each time instant is followed by exactly one next time instant. The truth of a temporal formula is determined by the truth values of its atomic propositions which may vary from time instant to time instant. The basic element of temporal logic language is a state formula P, which is any first-order logic formula. It is built from atomic predicates; the quantifiers $\exists$, $\forall$; the logical operators $\wedge$, $\vee$,$\neg$; and the "temporal" operators $\Box$("always"), $\Diamond$("eventually"), and $\circ$("next"), $\mathcal{U}$ ("until"), $\mathcal{W}$("weak until"), $\mathcal{R}$("release").

Linear temporal logic (LTL) is propositional temporal logic whose interpretations are limited to transitions which are discrete, reflexive, transitive, linear and total [26]. Linear-time temporal logic (LTL) has the following syntax given in Backus Naur form (Equation 2.1) [27]:

$$\begin{aligned}\phi ::== \top \mid \bot \mid p \mid (\neg)\phi \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid (\phi \Rightarrow \phi) \mid \\ (\Box\phi) \mid (\Diamond\phi) \mid (\circ\phi) \mid (\phi\mathcal{U}\phi) \mid (\phi\mathcal{W}\phi) \mid (\phi\mathcal{R}\phi)\end{aligned} \tag{2.1}$$

where ϕ is a formula and p is any propositional atom.

Let σ be a non-empty sequence of states, i.e. $\sigma = s_0,...,s_n$ where s_i is a state. A state that satisfies a state predicate P is called P-state. Let assume that P, P_1, P_2 are state predicates, temporal operators can be interpreted in Table 2.2.

Table 2.2: Meaning of temporal operators

Operator	**Meaning**
$\sigma \models \circ P$	σ is P-state in the next moment of time
$\sigma \models \Box P$	every state in σ are P-state.
$\sigma \models \Diamond P$	there exits some P-state in σ.
$\sigma \models P_1 \mathcal{U} P_2$	there exists some P_2-state s_k in σ and every state until s_k (excluding s_k) is P_1-state.

2.2 Classical set theory

Sets are fundamental concepts that can be used to define all other concepts in mathematics. The language of set theory is based on a single fundamental relation, called membership. a is said to be a member of B (denoted by $a \in B$), it means that B contains a as an element. A set which has no element is called an empty set or null set. An empty set is denoted by the symbol $\varnothing$ or $\{\}$. We recall some basic constructs of set theory as follows:

Set comprehension. Given any non-empty set s, we can define a new set by considering only those elements of s that satisfy some property p, i.e. $\{x \in s \mid p\}$.

Example: a set of person who owns a red car and has a address is expressed by $\{x \in Person \mid redcar(x), address(x)\}$.

Power set: If A is a set, then the set of all subsets of A is called the power set of A, denoted by $\mathbb{P}(A)$.

Example: Let A be a set $\{x, y\}$, $\mathbb{P}(A) = \{\varnothing, \{x\}, \{y\}, \{x, y\}\}$.

Ordered pair: Given two sets A and B, one of the basic constructions of set theory is the formation of an ordered pair, $\langle a, b\rangle$, where $a \in A$ and $b \in B$. The main property of ordered pairs is that if $\langle a_1, b_1\rangle$ and $\langle a_2, b_2\rangle$ are ordered pairs, where $a_1, a_2 \in A$ and $b_1, b_2 \in B$, then $\langle a_1, b_1\rangle$ $= \langle a2, b2\rangle$ iff $a_1 = a_2$ and $b_1 = b_2$.

Cartesian product: Given two sets A and B, the set of all ordered pairs $\langle a, b\rangle$, with $a \in A$ and $b \in B$, is a set denoted $A \times B$ and called the Cartesian product of A and B.

Relation: Given two sets A and B, a binary relation R between A and B is any set of ordered pairs from $A \times B$, i.e., $R \subseteq A \times B$.

The domain of the relation R is denoted by $dom(R)$ such that $dom(R) = \{a \in A \mid \exists\, b \in B, \langle a, b\rangle \in R\}$.

The range of the relation R is denoted by $ran(R)$ such that $ran(R) = \{b \in B \mid \exists\, a \in A, \langle a, b\rangle \in R\}$.

The relation R is functional if $\forall\, a \in A$ and $\forall\, b_1, b_2 \in B$, if $\langle a, b_1\rangle$ and $\langle a, b_2\rangle$, then $b_1 = b_2$.

Partial function: A partial function f from a set A to a set B, denoted by $f : A \nrightarrow B$ is a relation that does not contain two distinct pairs with the same first element. If $dom(f) = A$ then f is a total function.

Example: If the set of all people is *Person*, and the set of all locations is *Location*. We want to know the location of a person, then the information may be described by a relation $r \subseteq \mathit{Person} \times \mathit{Location}$. Moreover, at one time, a person can only at a place. Hence, it can be described by a partial function $where : Person \nrightarrow Location$.

2.3 Fuzzy sets and Fuzzy If-Then rules

2.3.1 Fuzzy sets

Many real-world software systems are developed from requirements of all stake holders. In fact, stake holders usually cannot describe the system precisely. They often use vague, ambiguous, fuzzy terms such as "very good", " far", "hot", etc. For example, a functionality of an air conditioner is described as "*When the outside temperature is hot, then air conditioner becomes cooler*". In order to deal with systems which are too complex or too ill-defined to admit of precise descriptions, Zadeh [28] introduced a logic framework which is not traditional two-valued, but multi-valued logics whose values are interpreted by Fuzzy sets.

Fuzzy sets are actually functions that map a value that might be a member of a set to a number between zero and one indicating its actual degree of membership. A fuzzy set F defined on an universal set X is a set of ordered pairs illustrated in Equation 2.2 [29].

$$F = \{(x, \mu_F(x))\} \tag{2.2}$$

where $x \in X$ and $\mu_F(x) : X \to [0, 1]$ is termed as the grade of membership of x in F.

Example: Let S be a set of all real numbers and let S_f is a set of positive and large numbers. Then $S_f = \{x \in S \mid x \text{ is positive and large}\}$.

We introduce a membership degree function to measure the fuzzy term "large" as follows:

$$\mu_{large}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 - e^{-x} & \text{if } x > 0 \end{cases} \tag{2.3}$$

Fuzzy sets use so-called linguistic variables in addition to numerical variables. The values of a linguistic variable are labels of fuzzy subsets of X which have the form of phrases or sentences in a natural or artificial language. For example, *height* is a linguistic variable labeled x, the values of x might be "tall", "not tall", "very tall", or "tall but not very tall". Generally, a value of a linguistic variable is a concatenation of atomic terms that can be divided into main categories shown below:

- primary terms: which are labels of specified fuzzy subsets of the universe set (for instance: *tall* in the above example).
- hedges: such as "very", "slightly", etc.
- negation and connectives symbols (i.e *not*, *and*, *or*).

A fuzzy hedge is an operator which transforms the fuzzy set $F(x)$ into the fuzzy set $F(hx)$. The hedges are the functions that generate a larger set of values for linguistic variables. For instance, using hedge *very* along with negation *not* applied to the term *tall*, we can have *very tall* or *not very tall.*

2.3.2 Fuzzy If-Then rules

In classical propositional logic, the expression **If A Then B**, denoted by $A \Rightarrow B$, where A and B are propositional variables. The implication operator is defined in the Table 2.3.

Table 2.3: Truth table of implication operator

A	**B**	**A $\Rightarrow$ B**
T	T	*T*
T	F	*F*
F	T	*T*
F	F	*T*

We continue to consider the implication the implication $a \Rightarrow b$, where $a \in A$,$b \in B$, and A, B are fuzzy sets. This implication can be represented as "IF $a \in A$ is true with a truth value $\mu_A(a)$ THEN $b \in B$ is true with a truth value $\mu_B(b)$". Then it is written in a simple form: "If a is A then b is B". Generally, we have n fuzzy sets: $A_1, ..., A_n$ and a fuzzy set B. Then the rule is defined as follows:

$$\textbf{IF } a_1 \text{ is } A_1 \textbf{ and } ... \textbf{ and } a_n \text{ is } A_n \textbf{ THEN } b \text{ is } B$$

Example: IF the weather is bad THEN the speed is slow.

Fuzzy If-Then rules play an important role in fuzzy sets. It provides an approach to analysing imprecise description of systems. We usually use these rules for describing the behavior of such systems.

2.4 Formal methods

One of important goals of software engineering is to enable developers to build a complex software system with reliability. Formal methods which can be used to specify and verify systems mathematically are one way to accomplish this goal. A method is formal if it has well-defined mathematics basis, typically given by a formal specification language [30].

Formal specification is a process that describes the system and its desired properties such as functional behaviour, timing behaviour, non-functional properties by a language with mathematically-defined syntax and semantics. When we make a formal specification, we need to make a detailed systems analysis that usually reveals errors and inconsistencies in the informal requirements specification. Two fundamental approaches to formal specification have been used to describe specifications including algebraic and model-based approaches.

- Algebraic approach: Systems are specified in terms of a sequence of actions and their relationship. The algebraic approach is particularly suitable for the definition of sub-system interfaces. This method of formal specification defines an object class or an abstract data type in terms of the relationships between the type operations. Several languages for algebraic specification have been developed including Larch [31] and OBJ [32].
- Model-based approach: Model-based specification is an approach to formal specification where the system specification is expressed as a system state model. This state model is constructed using well-understood mathematical entities such as sets and functions. System operations are specified by defining how they affect the state of the system model. The most widely used notations for developing model-based specifications are Z [33] and VDM [34] which focus on specifying sequential systems while CSP [35] and Petri Nets [36] concentrates on specifying concurrent ones.

Formal verification methods have recently become usable by industry and there is a growing demand for professionals able to apply them. Two well-established approaches to verification are model checking and theorem proving.

Model checking is a technique for verifying finite state concurrent systems such as sequential circuit designs and communication protocols. It has a number of advantages over traditional approaches that are based on simulation, testing, and deductive reasoning. In particular, model checking is automatic and usually quite fast. Also, if the design contains an error, model checking will produce a counterexample that can be used to find the source of the error. The main challenge in model checking is dealing with the state space explosion problem. This problem occurs in systems with many components that can interact with each other or systems with data structures that can assume many different values. In such cases the number of global states can be enormous.

Theorem proving is a technique where both the system and its desired properties are specified in mathematical logic formulas. It is a process which finds the proof of a property from axioms of the system. Let Γ is a set formulas of the system's description, ψ is a formula of the system's properties specification. Then this verification method tries to find the a proof $\Gamma \vdash \psi$.

The thesis uses Event-B formal method to model and verify event-driven systems. Hence, before introducing it we briefly present several different formal methods which inspire Event-B's ideas such as VDM, Z [33], B [14]. Z and B are both invented by J. Abrial and his colleagues.

2.4.1 VDM

VDM stands for "The Vienna Development Method" which a collection of techniques for the formal specification and development of computing systems. VDM is a model-based method giving descriptions of software systems and other systems as models [37]. Models are specified as objects and operations on objects, where the objects represent input, output, and internal state of the system. It consists of a specification language called VDM-SL (VDM-Specification Language); rules for data and operation refinement that allow one to establish links between abstract requirements specifications and detailed design specifications down to the level of code; and a proof theory in which rigorous arguments can be conducted about the properties of specified systems and the correctness of design decisions. VDM-SL is a model-oriented specification language. This means that a specification in VDM-SL consists of a mathematical model built from simple data types like sets, lists and mappings, along with operations which change the state of the model. VDM-SL has a formally defined semantics. The logic underlying this semantics is based on the Logic of Partial Functions (LPF).

A VDM-SL model is a system description given in terms of the functionality performed on data. It includes a collection of definitions of data types and functions or operations performed upon them. Several data types are defined in VDM-SL such as *bool*, *nat*, *nat1*, *int*, *rat*, *real*, *char*, *token*.

A traditional abstract VDM model usually contains the following components:

- Semantic domains: to define the involving objects.
- Invariants: to define a set of conditions to limit the set of operating objects defined by the semantic domains by defining a set of conditions.
- Syntactic domains: to define the syntax for manipulating the objects defined by the semantic domains.

- Semantic functions: to define the effect of commands on the objects defined by the semantic domains.

2.4.2 Z

The Z notation is based upon set theory and first-order predicate calculus. Every object in the mathematical language has a unique type, represented as a maximal set in the current specification. One aspect of Z is the use of natural language. It uses mathematics to state the problems, to discover solutions, and to prove that the chosen design meets the specification. Z provides refinement mechanism that allows to develop the system gradually. A Z specification document consists of interleaved passages of formal, mathematical text and informal explanation [33]. The formal text consists of a sequence of paragraphs which gradually introduce the schemas, axioms, constraints and basic types of the specification.

- Basic type declaration: A basic type definition introduces one or more basic types. These names must not have a previous global declaration, and their scope extends from the definition to the end of the specification. It has the form as follows:

  $$Paragraph ::= [Ident, ..., Ident]$$

  where *Paragraph* is declaration paragraph and *Ident* is a Word.

- Axiomatic descriptions: An axiomatic description introduces global variables which have not been declared before. While the part [*declaration*] is an acceptable form of axiom descriptions and is required, the part [*predicates*] is optional. If the part [*predicates*] is absent, then the default predicate is *true*. The axioms have the form as follows:

$$\begin{array}{|l} [declaration] \\ \hline [predicates] \end{array}$$

- Schemas: Schema specifications consists of three parts: schema name, declaration, and predicates. The part *schema name* which have not been declared before is the identifier of the schema. The part *declaration* contains one or more declarations and while the part *predicates* includes one or more predicates. A schema has the form as follows:

$$\begin{array}{|l} \multicolumn{1}{l}{SchemaName} \\ \hline D1;\ ...;\ Dn \\ \hline P1;\ ...;\ Pn \\ \hline \end{array}$$

where $D1...Dn$ are declarations and $P1...Pn$ are predicates.

2.4.3 B method

B is a formal method for specifying, designing, and coding software systems. The main idea of B is to start with a very abstract model of the system under development and gradually add details by building a sequence of more concrete models [14]. B provides the concept of an abstract machine which encapsulates a set of mathematical items, constants, sets, variables and a collection of operations on these variables. These elements are contained in a named module which can be seen or used in other modules. A general specification of an abstract machine can be written as follows:

```
MACHINE M(p)
CONSTRAINTS CONSTANTS k
PROPERTIES p
VARIABLES v
INVARIANT I
INITIALIZATION L
OPERATIONS
  y ← op(x)
       PRE P
       THEN
         S
       END
  ......
END
```

The machine clauses have the following syntax and semantics:

- *parameters*: The machine M can have a list of parameters p. These parameters are assumed to be independent of each other and their logical properties are specified by CONSTRAINTS clauses.
- *sets*: Each item defined in the sets clause is corresponding to a definition of given or enumerated sets.
- *constants*: this clause contains a list of identifiers which are constant within the operations of the machine.
- *properties*: The clause containing predicates and formula conjunction defines logical properties for sets and constants.

- *variables*: It consists of a list of identifiers which can be separated by comma. The clause introduces the variables (components) of the state of the machine.
- *invariant*: It define constraints of variables including the typing of the variables. It consists of predicates and formulas of variables, constants, sets and parameters of the machine.
- *operations*: The clauses modifying the state variables defines the dynamic part of the machine. The input and output parameters declaration has the form: $y \leftarrow \text{opname}(x)$.

Before-After predicates: These predicates are used to express variables state modification relating variables values at just *before* and *after* the operations take place. The $before-value$ and $after-value$ relate to each other by both deterministic and non-deterministic ways.

Substitutions: Substitutions describe the specifications of the operations with **BEGIN...END** structures. B method provides several kinds of substitutions as follows:

- Multiple simple substitutions have the form: $[x, y := E, F]\, P$ where x and y are two distinct variables, E and F two expressions and P a formula. It denotes P where free occurrences of x and y are replaced simultaneously by E and F respectively.
- Conditional substitution is defined by means of the **IF** ... **THEN** ... **ELSE** ... **END** construct.
- Bounded choice substitution gives a notation to express the choice between two substitutions S and T. It introduces a kind of bounded non-determinism.

Table 2.4: Comparison of B, Z and VDM [1]

Attributes	**Z**	**VDM**	**B**
Basis	Predicate Calculus, Set theory, Schema	Partial Functions Set Theory	Weakest Preconditions, Set Theory
Development Stages	Specification	Specification Design	Specification Design, Implementations
Style	Schema Notations, Relations	Pre/Post Conditions, Functions	Rigorous Programming Language
Tool Support	At specification level	At specification level	All development stages

Table 2.4 lists and compares the main attributes of Z, B and VDM.

There are several supporting tools for B method including open source and commercial licences. The B Toolkit [17] generates proof obligations and provides supporting tools for the automatic discharge of those proof obligations. It also supports for the generation of documentation, and for the browsing of developments. Atelier-B [38] is an industrial tool that allows for the operational use of the B Method. This tool has been used in various subways systems and automotive industry.

2.5 Event-B

2.5.1 An overview

Event-B is a formal method for system-level modeling and analysis and is an evolution of B-method. Key features of Event-B are the use of set theory as a modeling notation, the use of refinement to represent systems at different abstraction levels and the use of mathematical proof to verify consistency between refinement levels [13]. An Event B model encodes a state transition system where the variables represent the states and the events represent the transitions from one state to another. The structure of an Event B model is given in Figure 2.1 consisting of two basic constructs: machines and contexts. Contexts are static parts of a model, while *machines* specify the dynamic parts of a model. The relationship between a *machine* and a *context* is *see*, i.e., the *machine* can refer to elements defined in the *context*.

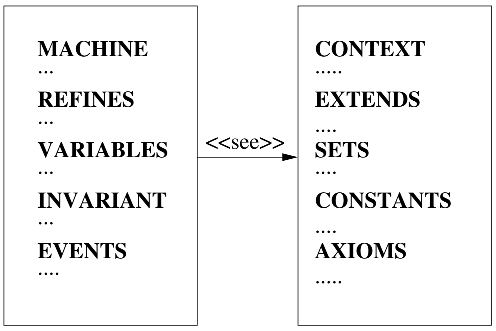


FIGURE 2.1: Basic structure of an Event B model

2.5.2 Event-B context

An Event-B context describes the static part where all the relevant properties and hypotheses are defined. Each context has an unique name which is different from others elements of Event-B. A context can extend another context, i.e. it can refer to sets of clauses defined in the extended context. A context generally consists of the following items:

- Sets are items which describe a set of abstract and enumerated types.
- Constants are clauses containing a list of constants introduced in a context. Each constant also has a distinct identifier.
- Axioms are a list of various predicates of constants in first-order logic expression. These predicates will be present as hypotheses of

all proof obligations. Types and constraints are described in this clause.

- Theorems are logical expressions which have to be proved from the axioms.

An example of a context is given in Figure 2.2 in which we have a context named *ctx_0* with a set PERSON and two constants n and fid, where n is a natural number and fid is a total function that maps from PERSON to a set of natural numbers.

```
CONTEXT  ctx_0
SETS
      PERSON
CONSTANTS
      n
      fid
AXIOMS
      axm1 : n ∈ ℕ
      axm2 : fid ∈ PERSON → ℕ
THEOREMS
      thm1 : n ≥ 0
END
```

FIGURE 2.2: An Event-B context example

2.5.3 Event-B Machine

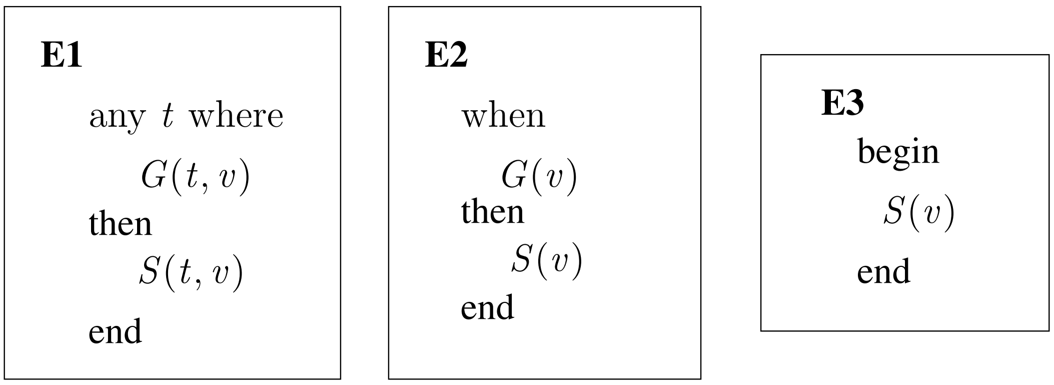
Sets and constants defined in a context can be used by a machine which is composed of a set of clauses. A machine can see a context and refine another machine. A general structure of an Event-B machine consists of items as follows:

- Variables represent the state of the model.

- Invariants are first order logic expressions describing the properties of the attributes defined in the *variables* clause. These properties are true in the whole model. Invariants need to be preserved by all events execution.

- Theorems define a set of logical expressions that can be deduced from the invariants. Unlike invariants, they do not need to be proved to be preserved by events.

- Events define all the events that occur in a given model. An Event-B event consists of a guard $G(t, v)$ and an action $A(t, v)$, where t is any parameter of the event, v is the variable. An event is fired when its guard $G(t, v)$ is evaluated to true. If several guards evaluate to true, only one is fired with a non deterministic choice. Action $A(t, v)$ describes how variables involve in the events. There are three forms of an Event-B event illustrated in Figure 2.3. An event E1 is a form with parameters, an event E2 without any parameter, and an event E3 without any guard.



```
E1
    any $t$ where
        $G(t, v)$
    then
        $S(t, v)$
    end
```

```
E2
    when
        $G(v)$
    then
        $S(v)$
    end
```

```
E3
    begin
        $S(v)$
    end
```

Figure 2.3: Forms of Event-B Events

The action of an event may be skip or composed of several assignments of the form.

$$x := E(t, v) \tag{2.4}$$

$$x :\in E(t, v) \tag{2.5}$$

$$x :\mid E(t, v) \tag{2.6}$$

The action assignment having the form 2.4 is deterministic which directly assigns $E(t, v)$ to x. Assignment in form 2.5 is non-deterministic which assigns an element of set $E(t, v)$ to x. Assignment in form 2.6 is a generalized form of assignment.

2.5.4 Event-B mathematical language

The basis for the formal models in Event-B is first-order logic and set theory. The first-order logic of Event-B contains standard operators such as *conjunction*($\wedge$), *disjunction*($\vee$), *implication*($\Rightarrow$), *equivalence*($\equiv$), *negation*($\neg$), *universal quantification*($\forall$), *existential quantification* ($\exists$).

Event-B mathematical language has an important part which is set-theoretical notation, with the introduction of the membership predicate $E \in S$, i.e. the expression E is a member of set S. A set can be defined by listing all its members, or constructed by Cartesian product, power set and set comprehension.

For example, given two sets S and T, the Cartesian product of S and T is stated $S \times T$ which is the set of ordered pairs $x \mapsto y$ where $x \in S$ and $y \in T$. The power set of S is denoted by $\mathbb{P}(S)$ is the set of all subsets of S. Set comprehension $\{x \in E \mid P\}$ is the set of elements of E such that P holds where E is an set and P is a predicate.

Event-B set-theoretical notation has a key feature which is the relations and functions. Given a relation r, two sets S and T, and two distinct variables a and b. The definition of some functions and relations in Event-B is illustrated in Table 2.5.

Table 2.5: Relations and functions in Event-B

Event-B construct	**Definition**
$r \subseteq S \leftrightarrow T$	$r \subseteq \mathbb{P}(S \times T)$
$dom(r)$	$\{a \in S \mid \exists\, b \in T, \langle a, b\rangle \in r\}$
$ran(r)$	$\{b \in T \mid \exists\, a \in S, \langle a, b\rangle \in r\}$

2.5.5 Refinement

To deal with complexity in modeling systems, Event-B provides a refinement mechanism that allows us to build the system gradually by adding more details to get more precise models. A context can be extended by at most another context. A concrete Event-B machine can refine at most one abstract machine. The sets and constants of an abstract context are kept in its refinement. In other words, the refinement of a context just consists of adding new carrier sets and new constants to existing sets and constants. These are defined by means of new properties. Refinement of context and machine in Event-B is illustrated in Figure 2.4.

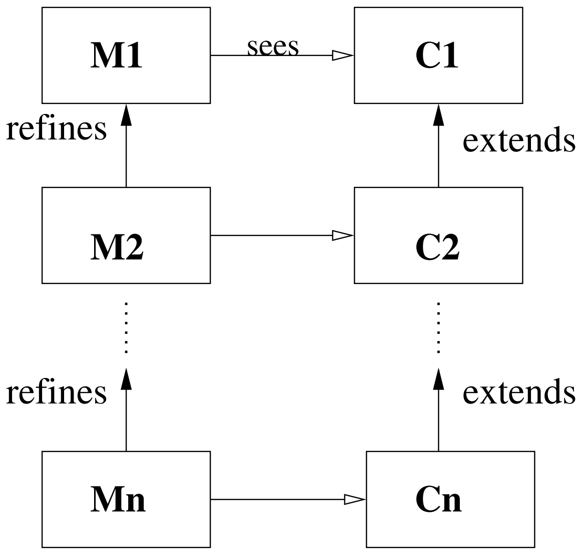


Figure 2.4: Event-B refinement

A refined machine usually has more variables than its abstraction as we have new variables to represent more details of the model. Event-B currently provides two types of refinement including superposition refinement and vertical refinement. In former, the abstract variables are retained in the concrete machine, with possibly some additional variables. The later is similar to data refinement where the abstract variables v are replaced by concrete ones w. Subsequently, the connections between them are represented by the relationship between v and w, i.e. gluing invariants $J(v, w)$.

Each event of an abstract machine is refined by one or more concrete events of refined machines. In Figure 2.5, event *ea* is refined by event *ec*.

```
ea
    any $t$ where
        $G(t, v)$
    then
        $S(t, v)$
    end
```

```
ec
    refines $ea$
    any $u$ where
        $H(u, w)$
    then
        $T(u, w)$
    end
```

FIGURE 2.5: Event refinement in Event-B

2.5.6 Proof obligations

In order to check if a machine satisfies a collection of specified properties, Event-B defines proof obligations (POs) which we must prove. They are automatically generated by the Rodin tool called proof obligation generators. This tool statically checks the syntax of the model including contexts and machines, then decides what to be proved. This outcome is then transferred to the provers for automatically or interactively proving. We will discuss some of the proof obligations which are relevant to the thesis such as: invariant preservation (INV), convergence (VAR), deadlock-freeness (DLF).

Invariant preservation proof obligation (INV PO) rule means that we must prove that invariants hold after event's execution. Let assume that we use an event *evt* defined as follows

```
evt
        any $x$ where
            $G(s, c, v, x)$
        then
            $v :| G(s, c, v, x)$
        end
```

The INV proof obligation of invariant *inv* of the event *ev* is named "evt/inv3/INV". The proof obligation is shown in Table 2.6.

Table 2.6: INV proof obligation

Axioms	$A(s, c)$	
Invariants	$I(s, c, v)$	
Guards	$G(s, c, v, x)$	evt/inv3/INV
Before-after predicates	BA(s,c,v,x,x')	
$\vdash$	$\vdash$	
Modified invariant	$I(s, c, v')$	

Variant proof obligation(VAR PO) rule ensures that each convergent event decreases the proposed numeric variant or proposed finite set variant. Let assume that a convergent is defined in Figure 2.6.

Then, its VAR PO is define as Table 2.7 and Table 2.8. The former is the case that variant $n(s, c, v)$ is numeric and the later is the case that $t(s, c, v)$ is a finite set.

Deadlock-freeness for a machine ensures that there are always some enabled events during its execution. Assume that a machine contains a set of n events $e_i(i \in \overline{1..n})$ of the following form: evt = **any** x **where**

```
evt
  status
      convergent
  any $x$ where
      $G(x, s, c, v)$
  then
      $v :| BA(s, c, v, x, v')$
  end
```

FIGURE 2.6: A convergent event

TABLE 2.7: VAR PO with numeric variant

Axioms	$A(s, c)$	
Invariants	$I(s, c, v)$	
Guards	$G(s, c, v, x)$	evt/VAR
Before-after predicates	BA(s,c,v,x,x’)	
$\vdash$	$\vdash$	
Modified variant is smaller	$n(s, c, v') < n(s, c, v)$	

TABLE 2.8: VAR PO with finite set variant

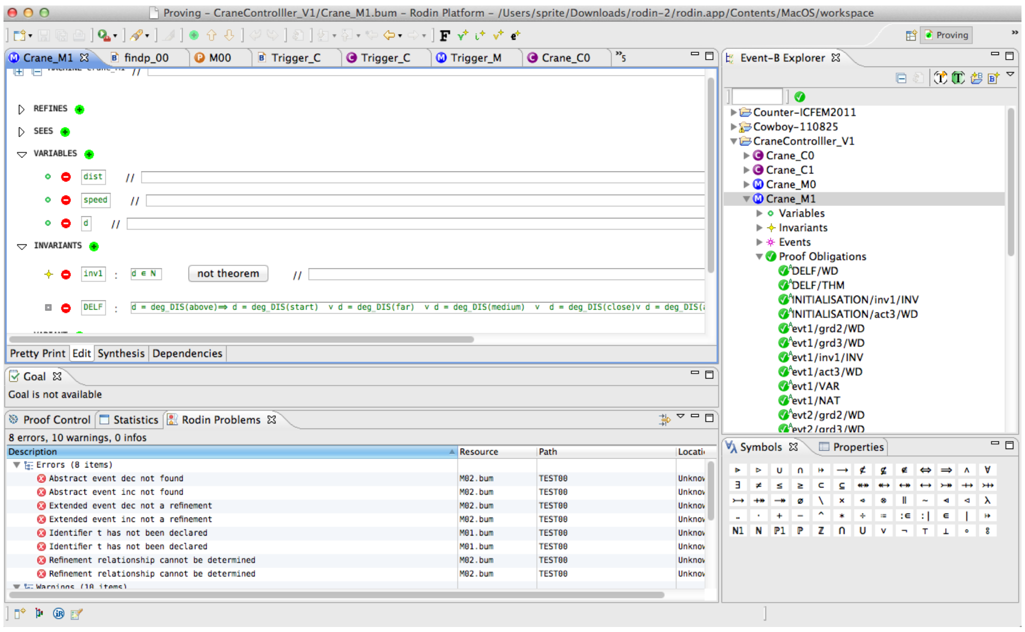
Axioms	$A(s, c)$	
Invariants	$I(s, c, v)$	
Guards	$G(s, c, v, x)$	evt/VAR
Before-after predicates	BA(s,c,v,x,x’)	
$\vdash$	$\vdash$	
Modified variant is smaller	$t(s, c, v') \subset t(s, c, v)$	

$G(x, v)$ **then** $A(x, v, v')$ **end**. The proof obligation rule for deadlock-freeness is illustrated in Equation 2.7.

$$I(v) \vdash \bigvee_{i=1}^{n} (\exists\, x_i . G(x_i, v)). \tag{2.7}$$

2.6 Rodin tool

This thesis uses The RODIN toolkit version 2.8 [16] which is an Eclipse environment for modeling and proving in Event-B. It is built on top the Eclipse platform and it contains a set of plug-ins used to support modeling and proving Event-B models. The Rodin tool provides a rich of perspective windows to user such as proving, Event-B editors, etc.(Figure 2.7).



Proving – CraneControlller_V1/Crane_M1.bum – Rodin Platform – /Users/sprite/Downloads/rodin-2/rodin.app/Contents/MacOS/workspace

```
REFINES
SEES
VARIABLES
    dist   //
    speed  //
    d      //
INVARIANTS
    inv1 : d ∈ N    not theorem   //
    DELF : d = deg_DIS(above)⇒ d = deg_DIS(start)  ∨ d = deg_DIS(far)  ∨ d = deg_DIS(medium)  ∨  d = deg_DIS(close)∨ d = deg_DIS(
```

Goal is not available

8 errors, 10 warnings, 0 infos

Description	Resource	Path	Locati
Errors (8 items)			
Abstract event dec not found	M02.bum	TEST00	Unknov
Abstract event inc not found	M02.bum	TEST00	Unknov
Extended event dec not a refinement	M02.bum	TEST00	Unknov
Extended event inc not a refinement	M02.bum	TEST00	Unknov
Identifier t has not been declared	M01.bum	TEST00	Unknov
Identifier t has not been declared	M01.bum	TEST00	Unknov
Refinement relationship cannot be determined	M02.bum	TEST00	Unknov
Refinement relationship cannot be determined	M02.bum	TEST00	Unknov
Warnings (10 items)			

```
Counter-ICFEM2011
Cowboy-110825
CraneControlller_V1
    Crane_C0
    Crane_C1
    Crane_M0
    Crane_M1
        Variables
        Invariants
        Events
        Proof Obligations
            DELF/WD
            DELF/THM
            INITIALISATION/inv1/INV
            INITIALISATION/act3/WD
            evt1/grd2/WD
            evt1/grd3/WD
            evt1/inv1/INV
            evt1/act3/WD
            evt1/VAR
            evt1/NAT
            evt2/grd2/WD
            evt2/grd3/WD
```

FIGURE 2.7: The Rodin tool

We present some important windows as follows:

- Proving perspective: It provides all proof obligations which are automatically generated for Event-B machines. These proof obligations can be discharged automatically or interactively with hypotheses and goal windows.

- Event-B perspective: This perspective includes windows which allows us to edit Event-B machines and contexts. If users encode incorrectly, problem windows will show the error's content.

2.7 Event-driven systems

Contrasting with centralized systems where control decisions are determined by the values of system state variables, event-driven systems are driven by externally generated events. There are many types of event-driven systems including many editors where user interface events signify editing commands, rule-based production systems which are used in AI where a condition becoming true causes an action to be triggered, and active objects where changing a value of an object's attribute triggers some actions [6]. In the thesis, we consider two later kinds of event-driven systems: context-aware systems and database triggers systems.

2.7.1 Event-driven architecture

In software systems, an event can be defined as a notable thing that happens inside or outside the system. The term event refers to both the specification and instances of events. An event usually has event header and event body. While the former contains event's description, the latter specify what happened.

In an event-driven system, its components cooperate by sending and receiving events. The sender delivers an event to an dispatcher. The event dispatcher is in charge of distributing the event to all the components that have declared their interest in receiving it. Thus, the event dispatcher allows decoupling between the sources and the recipients of an event. A common event-driven system has three general styles of event processing: simple, stream, and complex ones. It may consist of some main parts such as event processing, event tooling, source and target, event meta-data.

Event-driven architecture allows to build applications and systems which are more responsive. It also allows to develop and to maintenance the large-scale, distributed software systems involving unpredictable occurrences.

In this thesis, we consider two applications of active objects and rule-based production systems: active databases and context-aware systems. Both systems use form of Event-Condition-Action (ECA) rules to describe their behavior. Besides this common characteristic, each system has specific and interesting properties need to be analyzed.

2.7.2 Database systems and database triggers

Database management system can be classified broadly into two types as follows:

- In passive database management systems, users query the current database to retrieve the corresponding data of these queries. Traditional database management systems are passive because they passively wait for actions from users then response. After returning data for queries, they wait again for the next queries.
- Active database management systems are data-driven or event-driven systems. They use active rules to react to database changes. A set of active database rules defines reactive behavior of the active database.

A relational database system which is based on the relational model consists of collections of objects and relation, operations for manipulation and data integrity for accuracy and consistency. Modern relational database systems include active rules as database triggers responding to events occurring inside and outside of the database.

Database trigger is a block code that is automatically fired in response to an defined event in the database. The event is related to a specific data manipulation of the database such as inserting, deleting or updating a row of a table. Triggers are commonly used in some cases: to audit the process, to automatically perform an action, to implement complex business rules. The structure of a trigger follows ECA structure, hence it takes the following form:

rule_name:: Event(e) IF condition DO action

It means that whenever *Event*(e) occurs and the *condition* is met then the database system performs *actions*. Database triggers can be mainly classified by two kinds: Data Manipulation Language(DML) and Data Definition Language (DDL) triggers. The former is executed when data is manipulated, while in some database systems, the latter is fired in response to DDL events such as creating table or events such as login, commit, roll-back, etc.

Users of some relational database systems such as Oracle, MySQL, SyBase are familiar with triggers which are represented in SQL:1999 format [39] (the former is SQL-3 standard). The definition of SQL:1999 trigger has the syntax as follows:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>

    {BEFORE|AFTER} {INSERT|DELETE|UPDATE}
    ON <table_name>

    [REFERENCING [NEW AS <new_row_name>]
    [OLD AS<old_row_name>]]

    [FOR EACH ROW
       [WHEN (<trigger_condition>)] ]

    <trigger_body>
```

2.7.3 Context-aware systems

The term "context-aware" was first introduced by Bill Schilit [40], he defined contexts as location, identities of objects and changes of those objects to applications that then adapt themselves to the context. Many works have been focused on defining terms of context awareness. Abowd *et al.* [41] defined a context aware system is a system that has the ability to detect and sense, interpret and respond to aspects of a user's local environment and to the computing devices themselves. Context-aware systems can be constructed in various methods which depend on requirements and conditions of sensors, the amount of users, the resource available on the devices. A context model defines and stores context data in a form that machines can process. Baldauf *et al.* [42] summarized several most relevant context modeling approaches such as key-value, markup scheme, graphical object oriented, logic based and ontology based models. Chen [2] also defined three different approaches to achieving contextual data as follows:

- Direct sensor access: The systems directly gather contextual data from built-in sensors. This approach does not require any additional layer but it is just suitable for simple cases not for distributed systems.
- Middle-ware infrastructure: It introduced layered architecture to separate business logic and user interfaces of the system. The system is extensible because it does not have to modify if sensors access changes.
- Context server: It allows multiple clients to use same resources. This approach is based on client-server architecture. Collected contextual data is stored in a so-called context-server. Clients use appropriate network protocols to access and use this data.

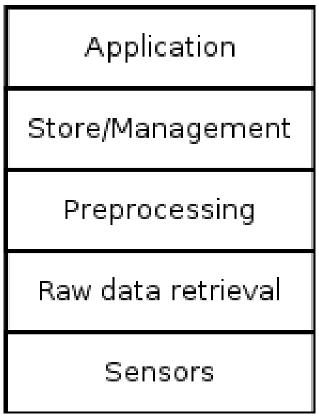


Figure 2.8: A layered conceptual framework for context-aware systems [2]

Figure 2.8 illustrates a layered conceptual framework for context-aware systems. The first layer consists of physical or virtual sensors which are able to capture context data from the environment. The second layer is able to retrieve data from sensors using providing API. Before storing the data in the fourth layer, it can be preprocessed in the third layer which is responsible for reasoning and interpreting contextual information. Finally, the application layer actually implementing reactions to different events which are raised by context changes.

In this thesis, we focus on a context-aware system which directly uses contextual data from physical sensors. The system senses many kinds of contexts in its working environment such as position, acceleration of the vehicle and/or temperature, weather, humidity, etc.. Processing of the system is context-dependent, i.e., it react to the context changes (for example: if the temperature is decreased, then the system starts heating). The system's behavior must comply with the context constraints properties (for instance: the system does not start heating, even though the operator executes heating function when the temperature is very high).

2.8 Chapter conclusions

In this chapter, we provided necessary backgrounds for the thesis. Section 2.2 provides basic knowledge about set theory which is one of the mathematic foundations of formal methods mentioned in the thesis. Section 2.1 and 2.3 are backgrounds of Chapter 5. Temporal logic can be used for presenting the liveness properties while Fuzzy sets and Fuzzy If-Then rules are used for describing the behavior of the imprecise system requirements. Section 2.4 gives an overview of formal verification and three formal methods such as VDM, Z, and B. The common point of these methods is that they use classical set theory as their basis. These ones also have the same approach of model-based specification. Section 2.5 presents about Event-B which is an evolution of B method in detail, because it is the formal language used for proposing methods of the thesis. Section 2.7 introduces briefly about event-driven architecture and its two domain applications such as database systems and context-aware systems which will be referred in Chapter 3 and Chapter 4 respectively.

In the next chapter, we present the first research result of the thesis on modeling and verifying database trigger systems using Event-B.

Chapter 3

Modeling and verifying database trigger systems

3.1 Introduction

Nowadays, many database applications are developed and used in many fields of modern life. Traditional database management systems (DBMSs) are passive as the database executes commands when applications or users perform appropriate queries. The research community has rapidly realized the requirement for database system to react to data changes. The event-driven architecture has been applied in active database systems to monitor and react to specific events happened in and outside the system. One of popular approaches which are used for specifying reactive semantics is Event-Condition-Action (ECA) rules. ECA rules have three parts: *event*, *condition*, and *action* parts. The *event* part describes the event happening inside or outside the system that the rule will handle. The *condition* part of the rule specifies the situation where the event occurs. The *action* part describes tasks which are performed if the corresponding event in the first part and the second part of the rule are evaluated to true.

Most of modern relational databases include these features in the form of database triggers. They use triggers to implement automatic task when a predefined event occurs. Triggers have two kinds: data manipulation language (DML) and system triggers. The former are fired whenever the DML statements such as *deleting*, *updating*, *insert* statements are executed, while the latter are executed in case that system or data definition language (DDL) events occur. A trigger is made of a block of code and has a syntax, for example, an Oracle trigger is similar to a stored procedure containing blocks of PL/SQL code. Trigger codes are human readable and does not have any formal semantic. Therefore, we can only check if a trigger conflicts to data constraints or leads to a infinite loop after executing it or with human inspection step by step. Hence, research work on a formal framework for modeling and verifying database triggers is desirable. Moreover, it is valuable if we can show that triggers execution is correct at early stage because it reduces the cost of database application development.

Several work have attempted to investigate in this topic by using termination detection algorithms or model checking [43, 44, 45, 46, 47]. However, most of work focused on the termination property, while few of them addressed to data constraints of the database system. A trigger is terminated but it still can cause critical problems if it violates data constraints. Furthermore, these approaches seem so complicated that we can not easily apply in the real database development.

In this chapter, we propose a new method to formalize and verify database triggers system using Event-B at early design phase. The main idea of the method comes from the similar structure and working mechanism of

Event-B events and database triggers. First, we propose a set of translation rules to translate a database system including triggers to an Event-B model. In the next step, we can formally check if the system satisfies data constraints preservation and find critical errors such as infinite loops by proving the proof obligations of the translated Event-B model. The advantage of our method is that a real database system including triggers and constraints can be modeled naturally by Event-B constructs such as *invariants* and *events*. The method also reuses Event-B proof obligations to prove such important properties of the systems. Therefore, the correctness of the entire system can be achieved mathematically and errors can be found by formal proofs. It is valuable especially for database application development since we are able to ensure that the trigger systems avoid the critical issues at the design time. With the supporting tool Rodin, almost proofs are discharged automatically, hence it reduces complexity in comparison to manual proving. We implement a tool called Trigger2B following the main idea to transform a database trigger model to a partial Event-B model automatically. It makes sense as we can bring the formal verification to database implementation. It also overcomes one of disadvantages that makes formal methods absent in the database development process because of the modeling complexity.

The remainder of this chapter is organized as follows: Section 3.2 summarizes the related research work. In Section 3.3, we present our method to model and check database systems including triggers. Section 3.4 introduces an extracted scenario of a human resource management application to demonstrate the model in detail. Section 3.6 concludes this chapter.

3.2 Related work

Many research work have been proposed for active rules or triggers verification. From the beginning, the work mainly focused on the termination of the triggers by using static analysis, e.g. checking set of triggers is acyclic with triggering graphs. In [43, 44], Sin-Yeung Lee and Tok-Wang introduced algorithms to detect the correctness of updating triggers. However, this approach was not able to be extended apparently for general triggers and it was presented as their future work. Our proposed method is able to handle with update, insert, and delete triggers. Furthermore, it also can check data constraint property.

E.Baralis [48] improved existing techniques to statically check if active rules are terminated or confluent. This approach is based on relational algebra that can be applied widely for active database rule languages and for trigger language (SQL:1999). In comparison to our proposed method, it does not consider data constraint property.

L. Chavarria and Xiaoou Li [46] proposed a method to verify active rules by using conditional colored Petri nets. Since Petri nets are mainly used in modeling transitions, it is quite elaborated when normalizing rules. The approach has to classify rules by their logic conditions to check if they involve disjunction or conjunction operators. In our opinion, if the number of these operators are enormous then the transition states can be exploded. The rule normalizing process also prevent database engineers to apply the method in the development.

Some work applied model checking techniques for active database rule analysis. In [49], T. S. Ghazi and M. Huth presented an abstract modeling framework for active database management systems and implemented a prototype of a Promela code generator. However, they did not describe how to model data and data manipulation actions for evaluation.

Eun-Hye CHOI *et al.* [50] proposed a general framework for modeling active database systems and rules. The framework is feasible by using a model checking tool, e.g SPIN, however, constructing a model in order to verify the termination and safety properties is not a simple step and can not be done automatically.

More recently, R. Manicka Chezian and T.Devi [51] introduced a new algorithm which does not pose any limitation on the number of rules but it only emphasizes on algorithms detecting the termination of the system. This approach, however, just checks the termination property.

In comparison to these approaches and methods, our proposed method is more suitable for database trigger systems. It is also such practical such that the tool Trigger2B can automatically translate a database system with simple triggers to Event-B models. It helps to reduce the complexity in system modeling.

3.3 Modeling and verifying database triggers system

As stated above, it is interesting to know whether a database system is designed correctly. Formal modeling of the system helps us not only to have a better understanding of the system but also enable us to verify the system's correctness and to resolve errors. In this section, we introduce a new method to model and verify a database system including triggers. This method allows to detect infinite loops and provides a warranty for data constraints preservation.

3.3.1 Modeling database systems

A database system is normally designed by several elements such as tables (or views) with integrity constraints and triggers. Whenever users modify the database table contents, i.e., executing Insert, Delete and Update statements, this data modification can fire the corresponding triggers and should be conformed to constraints. Before modeling a database system by Event-B, we introduce some database definitions in set theory which are the basis for modeling process.

Definition 3.1 (Database system)**.** A database system is modeled by a 3-tuple $db = \langle T, C, G \rangle$, where T is a set of table, C states system constraints, and G indicates a collection of triggers.

Definition 3.2 (Table)**.** For each $t \in T$, denoted by a tuple $t = \langle r_1, .., r_m \rangle$, where m is the total number of rows in the table t, r_i is a set indicating the i-th row of the table, $(i \in \overline{1..m})$. A row is stated by a tuple $r_i = \langle f_{i1}, .., f_{in} \rangle$, where n is total number of columns, f_{ij} represents data of column j at row i and $j \in \overline{1..m}$.

Definition 3.3 (Trigger)**.** Each trigger $g, g \in G$ of the system is presented as a 3-tuple $g = \langle e, c, a \rangle$, where e is type of the trigger's event, c is condition of the trigger, and a is the action of the trigger.

Based upon these definitions, we present a set of translation rules to translate a database model to an Event-B model illustrated in Table 3.1. These rules are described in detail as follows:

- Rule 1. A database system is formalized by a pair of Event-B machine and context: $\langle DB\_M, DB\_C \rangle$.
- Rule 2. A table is presented by a Cartesian product of N sets $T = \{TYPE_1 \times TYPE_2 \times .. \times TYPE_n\}$, where $TYPE_i$ denotes data type of column i.

- Rule 3. For each table T, we add a variable t such that $t \in \mathbb{P}(T)$ to the machine DB_M.
- Rule 4. Each table T has a primary key constraint. We encode this kind of constraints as a bijective function $f : TYPE_1 \twoheadrightarrow\!\!\!\!\!\!\!\!\!\rightarrowtail (TYPE_2 \times .. \times TYPE_n)$, we assume that the first column of the table is the primary key.
- Rule 5. A data constraint C is formalized by a invariant $\mathcal{I}$.
- Rule 6. A trigger E is translated to an event Evt.

The translation rules are summarized in Table 3.1.

TABLE 3.1: Translation rules between database and Event-B

	Database definitions	**Event-B concepts**
Rule 1.	$db = \langle T, C, G \rangle$	DB_M, DB_C
Rule 2,3	$r_i = \langle f_{i1}, .., f_{in} \rangle$ $t = \langle r_1, .., r_m \rangle$	$t \in \mathbb{P}(T)$ $T = TYPE_1 \times TYPE_2 \times .. \times TYPE_n$
Rule 4	Primary key constraint	$f : TYPE_1 \twoheadrightarrow\!\!\!\!\!\!\!\!\!\rightarrowtail TYPE_2 \times .. \times TYPE_n$
Rule 5	Constraint C	Invariant $\mathcal{I}$
Rule 6	Trigger E	Event Evt

Example: Let assume that a database system consists of two tables $T1$, $T2$ (both of them have two columns), two triggers $G1$, $G2$ and one data constraints C. The Event-B specification of the system is partially described in Figure 3.1.

3.3.2 Formalizing triggers

In this Section, we show in detail how to formalize database triggers. Recall that, a trigger is denoted by 3-tuple $g = \langle e, c, a \rangle$, where e is type of the trigger, c is condition in which the trigger happens, a is trigger's actions. As illustrated in Table 3.2, a trigger is translated to an Event-B

CONTEXT *DB_C*
CONSTANTS
  T1
  T2
AXIOMS
  axm1 : $\mathtt{T1} = \mathtt{TYPE1} \times \mathtt{TYPE2}$
  axm2 : $\mathtt{T2} = \mathtt{TYPE3} \times \mathtt{TYPE4}$
END

MACHINE *DB_M*
SEES *DB_C*
VARIABLES
  t_1
  t_2
  f_1
  f_2
INVARIANTS
  inv1 : $\mathtt{t}_1 \in \mathbb{P}(\mathtt{T1})$
  inv2 : $\mathtt{t}_2 \in \mathbb{P}(\mathtt{T2})$
  inv3 : $\mathtt{f}_1 \in \mathtt{TYPE1} \twoheadrightarrow\!\!\!\!\!\!\!+ \mathtt{TYPE}_2$
  inv3 : $\mathtt{f}_2 \in \mathtt{TYPE3} \twoheadrightarrow\!\!\!\!\!\!\!+ \mathtt{TYPE}_4$
  inv4 : $\mathcal{I}$
EVENTS
Event $G_1 \triangleq$
  ...
Event $G_2 \triangleq$
  ...
END

FIGURE 3.1: Partial Event-B specification for a database system

event where conjunction of trigger's type and its condition is the guard of the event. The actions of the trigger are translated to the body part of an Event-B event.

TABLE 3.2: Formalizing a trigger

IF (e) ON (c)	WHEN $(e \wedge condition)$
ACTION (a)	THEN (a) END

In this chapter, we focus on modeling DML triggers, i.e. triggers are fired when executing DML statements such as *delete*, *insert*, *update*. We represent the type of such statements by an Event-B variable *type*, for example: $type = \{update\}$ indicating that this trigger is fired when a *update* statement on a specific table is executed.

A trigger action is a block code and its syntax depends on database management systems. This block code also contains SQL statements. In order to show how our method works, we simplify the case by considering that the Action part of a trigger contains a sequence of DML statements without branch or loop statements. Hence, the action of a trigger consists of a sequence of *Insert*, *Update* or *Delete* statement. In case of Update and Delete triggers, the action statement contains conditions showing which rows are affected. Therefore, we add these conditions to guard of the translated event. More precisely, the mapping rules of each kind of statements are presented as follows:

- Insert: This statement has the form: "Insert into T values ($val1$,.., $valn$)" where $val1$,..$valn$ are values of the new record. We encode the input values as a parameter of the event, r, $r \in T$. More specifically, the translated event has the form *Evt*= **Any** r **Where** $(r \in T) \wedge e \wedge c$ **Then** $t := t \cup r$.
- Delete: This statement is generally written in the form: " Delete from T where $column1 = some\_value$". It will delete the record that has the first column's value is equal to $some\_value$. We add a parameter for the event representing the value $some\_value$. The event is specified in detail as follows *Evt*= **Any** v **where** $(v \in TYPE_1) \wedge e \wedge c$ **Then** $t := t - f(v)$.
- Update: The general syntax of this statement is "Update T set $column1 = value1$ where $column1 = some\_value$". This statement will update a record where value of the first column is equal to $some\_value$, similar to the case of delete statement, we encode the input values as parameters of the event. The description of the translated event is as follows: *Evt*= **Any** $v1, v2$ **where** $v1 \in TYPE_1 \wedge v2 \in TYPE_2 \wedge e \wedge c$ **Then** $t := \{1 \mapsto v1, 2 \mapsto v2\} \oplus t$.

The translation rules are summarized in Table 3.3.

TABLE 3.3: Encoding trigger actions

INSERT INTO T
VALUES (value1,..,valuen)
	ANY r
	WHEN $(r \in T \land e \land c)$
	THEN $T := T \cup r$
	END

DELETE FROM T
WHERE $\langle column1 = some\_value \rangle$
	ANY v
	WHEN $(v \in TYPE_1 \land e \land c)$
	THEN $t := t - \{v \mapsto f(v)\}$
	END

UPDATE T
SET column2=value
WHERE $\langle column1 = v_1 \rangle$
	ANY $v1, v2$
	WHEN $v1 \in TYPE_1 \land v2 \in TYPE_2 \land e \land c$
	THEN $t := \{1 \mapsto v_1, 2 \mapsto v_2\} \oplus t$
	END

3.3.3 Verifying system properties

After the transformation, taking advantages of Event-B method and its support tool, we are able to verify some properties of the database system model as follows:

- Infinite loop: Since a trigger can fire the other triggers, hence it probably leads to a infinite loop. This situation occurs when after a sequence of events, state of the system does not change. There are two ways to check this property of the system. The first one is to check deadlock-freeness (DLKF) proof obligation of Event-B which states that the disjunction of the event guards always hold under the properties of the constant and the invariant. The deadlock freedom rule is stated as $I(v), P(c) \vdash G_1(v) \lor ... \lor G_n(v)$ where v is variable, $I(v)$ denotes invariant, and $G_i(v)$ presents guard of the event. At the moment, the DLKF proof obligation is not generated automatically by the Rodin tool yet. However, we can generate it manually by adding a theorem saying the disjunction of guards. In some cases, DLKF theorem can not be deduced from a set of invariant $I(v)$ and constant predicates. We will prove that there is always at least one event executes at a time by showing that the disjunction of the events' guards are always true before and after event execution including *INITIALISATION* event.

- Constraint preservation: Data constraints are rules that the system should conform. In Section 3.3, data constraints are expressed by invariants, and triggers are formalized by events. We need to prove that a trigger does not break these rules by showing that they are preserved before and after executing the event. It is formally defined as $I(v), G(t,v), S(t,v,v') \vdash I(v')$. This is also the INV proof obligation of Event-B machine. Consequently, if INV PO is discharged then the constraint preservation is satisfied.

3.4 A case study: Human resources management application

In this section, we illustrate our proposed method on the extracted scenario of a human resources management application. We first describe the scenario and its designed database, after that we translate the database into an Event-B machine and verify the trigger execution.

3.4.1 Scenario description

Let assume that we have a database system of a human resource application which includes two tables EMPLOYEES and BONUS structured in Table 3.4.

The database system has a constraint: *The bonus of an employee with a level greater than 5 is at least 10.*

Table 3.4: Table EMPLOYEES and BONUS

EMPLOYEES		BONUS	
E_Id	level	E_Id	amount
0911	2	0911	2
0912	2	0912	2
0913	4	0913	4

It includes two triggers doing the following tasks:

Trigger 1. Whenever the level of employee is updated, his bonus is increased by 10

Trigger 2. If the employee's bonus is updated, then his level is increased by 1.

These two triggers are rewritten in the format of PL/SQL as follows:

```
 CREATE TRIGGER Trigger_1 BEFORE UPDATE
    OF level ON employees
    FOR EACH ROW
    BEGIN
       UPDATE bonus SET bonus.amount
              =bonus.amount  + 10
       WHERE bonus.E_id = employees.E_id;
    END IF;
END
```

```
CREATE TRIGGER Trigger_2 BEFORE UPDATE
   OF amount ON bonus
   FOR EACH ROW
   BEGIN
    UPDATE employees SET
      employees.level = employees.level+1
    WHERE bonus.E_id = employees.E_id;
 END
```

3.4.2 Scenario modeling

We apply the method presented in Section 3.3 for modeling the system as follows:

- Apply Rule 1: the database system is formalized by a context *Trigger_C* and a machine *Trigger_M*, where *Trigger_C* contains a set *TYPE* representing all kinds of trigger.
- Apply Rule 2: two constant sets *TBL_EMPL*, *TBL_BONUS* representing for two table *employee* and *bonus*. Each table has two columns, hence each set is a Cartesian product of two sets of natural numbers (Figure 3.2). Variables *bonus* and *empl* are added into the machine *Trigger_M*.
- Apply Rule 3: two variables *bonus_rec* and *empl_rec* denotes a row of the table *bonus* and *employee* respectively.
- Apply Rule 4: two bijective functions *pk_bonus*, *pk_empl* represent primary key relationship of table *bonus* and *employee* respectively.
- Apply Rule 5: The system constraint is formalized by invariant *SYS_CTR*. The specification of the machine is illustrated partly in Figure 3.3.

```
CONTEXT  TRIGGER_C
SETS
     TYPES
     TABLE_NAMES
CONSTANTS
     TBL_EMPL
     TBL_BONUS
AXIOMS
     axm1 : partition(TYPES, {insert}, {update}, {delete})
     axm2 : TBL_EMPL = ℕ × ℕ
     axm3 : TBL_BONUS = ℕ × ℕ
     axm4 : partition(TABLE_NAMES, {EMPL}, {BONUS})
END
```

Figure 3.2: A part of Event-B Context

```
MACHINE   DB_M
SEES   TRIGGER_C
VARIABLES
        bonus
        empl
        pk_bonus
        pk_empl
        type
        table
INVARIANTS
        inv1 : bonus ∈ ℙ(TBL_BONUS)
        inv2 : empl ∈ ℙ(TBL_EMPL)
        inv3 : type ∈ TYPES
        inv4 : pk_bonus ∈ ℕ ⇸ ℕ
        inv5 : pk_empl ∈ ℕ ⇸ ℕ
        SYS_CTR : ∀eid.eid ∈ dom(empl) ∧ pk_empl(eid) > 5 ⇒
               pk_bonus(eid) > 5
        INF_LOOP : (type = update ∧ table = BONUS) ∨ (type = update ∧
               table = EMPL)
END
```

FIGURE 3.3: A part of Event-B machine

Next, we formalize two triggers of the system as the method presented in Section 3.3.2. In this example, the triggers containing *update* statement are specified in Figure 3.4.

3.4.3 Checking properties

- Constraint preservation: Since the constraint property of the system is modeled by the invariant

  $$SYS\_CTR : \forall\, eid.eid \in dom(empl) \wedge pk\_empl(eid) > 5 \Rightarrow pk\_bonus(eid) > 10.$$

  We need to prove that the invariant is maintained before and after events execution. The proof obligation of *trigger*1 is illustrated in Table 3.5. Two events *Trigger*1 and *Trigger*2 of the machine *DB_M*

```
Event  trigger1 ≙
    any
        eid
    when
        grd1 : type = update
        grd2 : table = EMPL
        grd3 : eid ∈ dom(empl)
    then
        act1 : type := update
        act3 : table := BONUS
        act5 : bonus := {eid ↦ (pk_bonus(eid) + 10)} ⊕ bonus
        act5 : pk_bonus(eid) := pk_bonus(eid) + 10
    end
Event  trigger2 ≙
    any
        eid
    when
        grd1 : type = update
        grd2 : table = BONUS
        grd3 : pk_bonus(eid) ≥ 10
    then
        act1 : type := update
        act2 : table := EMPL
        act3 : empl := {eid ↦ (pk_empl(eid) + 1)} ⊕ empl
    end
```

FIGURE 3.4: Encoding trigger

generate two proof obligations called trigger1/SYS_CTR/INV, trigger2/SYS_CTR/INV respectively.

TABLE 3.5: INV PO of event *trigger*1.

$\forall nid.nid \in dom(empl\_rec) \wedge pk\_empl(nid) > 5 \Rightarrow pk\_bonus(nid) > 10$ $emplid \in dom(empl)$ $type = update$ $table = EMPL$ $\vdash$ $\forall nid.nid \in dom(empl\_rec) \wedge pk\_empl(nid) > 5$ $\Rightarrow (pk\_bonus \oplus \{emplid \mapsto pk\_bonus(emplid) + 10\})(nid) > 10$	trigger1/ SYS_CTR/ INV

These proof obligations are also automatically discharged in the Rodin tool. Hence, the system constraint is satisfied by two triggers.

- Infinite loop: As we proposed in Section 3.3.3, the invariant *INF_LOOP* which is the disjunction of the event' guards id added to the target machine. If we show that this invariant is preserved by machine DB_M, then the execution of two triggers leads to a infinite loop. The proof clause of the event $trigger1$ is presented in Table 3.6.

TABLE 3.6: Infinite loop proof obligation of event $trigger1$

$\forall nid.(nid \in dom(empl) \wedge$ $type = update \wedge table = BONUS \wedge$ $pk\_bonus(nid) \geqslant 10) \vee (type = update \wedge table = EMPL)) \wedge$ $emplid \in dom(bonus)$ $table = BONUS \wedge pk\_bonus(emplid) \geqslant 10$ $\vdash$ $\forall nid.(nid \in dom(\{emplid \mapsto pk\_empl(emplid) + 1\} \oplus empl) \wedge$ $update = update \wedge EMPL = BONUS \wedge$ $pk\_bonus(nid) \geqslant 10) \vee$ $(update = update \wedge EMPL = EMPL)$	$trigger1$ $/INF\_LOOP$ $/INV$

Two INV proof obligations are also generated and discharged automatically in the Rodin, i.e. the invariant clause is also proved to be preserved through events. Hence, two triggers lead to a infinite loop.

3.5 Support tool: Trigger2B

In this section, we present a tool named Trigger2B which supports for automatic translation from a database including triggers to an Event-B model. First, we present the architecture of the tool. After that, we shows how the main components of the tool are implemented.

3.5.1 Architecture

Following the method presented in Section 3.3, we implement a tool called Trigger2B to support designing and modeling a database system including trigger with Event-B. This tool can generate multiples XML-based format output in order to use in verification phase with supporting tools of Event-B such as Rodin, AterlierB. The architecture of this tool is illustrated in Figure 3.5.

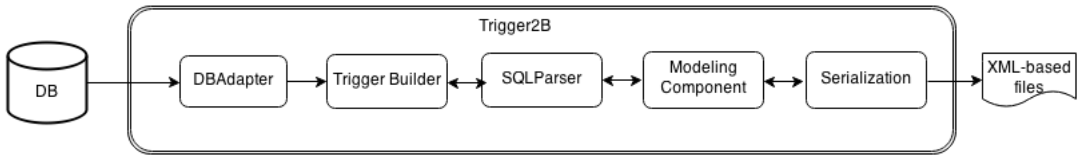


FIGURE 3.5: Architecture of Trigger2B tool

Main components of Trigger2B tool work as follows:

- **DBAdapter**: Manipulates relational database systems to get information about the database which will be modeled such as existing tables and triggers.
- **Trigger Builder**: Allows users to create new triggers based on the chosen database.
- **SQLParser**: Parses the trigger body to extract necessary elements, e.g. type and table names of SQL statements, for modeling.
- **Modeling Component**: Performs some algorithms to build a corresponding Event-B model.
- **Serialization**: Serialize the translated Event-B model to XML-based files such as Rodin Event-B components files.

3.5.2 Implementation

The heart of this tool is the modeling component which includes algorithms following the proposed translation rules to translate database concepts to Event-B constructs. The input of this component is the out

out of SQLParser component which currently uses ANLTR [52] framework to parse sql statements into a syntax tree. The parsed tree of general triggers is partially illustrated in Figure 3.6.

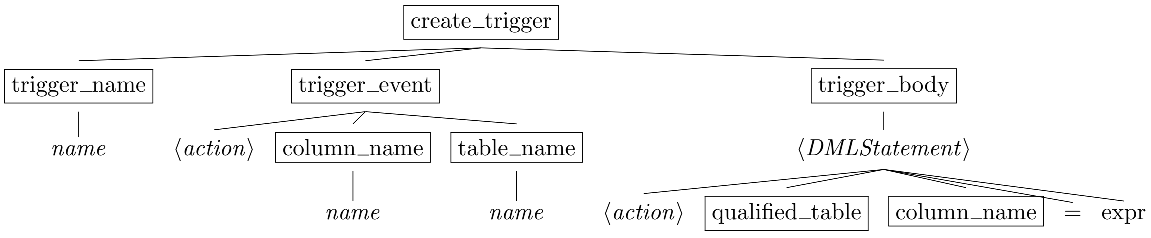


FIGURE 3.6: A partial parsed tree syntax of a general trigger

We propose an algorithm following our proposed translation rules to transform the parsed tree to an Event-B model. The algorithm is illustrated in Algorithm 1.

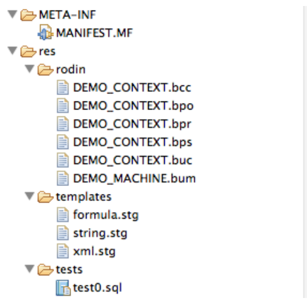
Alg. 1 TriggerModeling(t). An algorithm for translating a parsed tree t to an Event-B model

Input: Parsed syntax tree(t)
Output: Event-B machine (M)

```
begin
    node = root(t)
    while (isVisited(node)=false)
        if node.type = create_trigger then
            e=createNewEvent(M)
            if node.type = trigger_name then
                e.name = node.name
            elseif node.type = trigger_event
                for child in nodes.childs
                    if node.type = action then
                        addGuard(e,type=node.value)
                    if node.type = tabletable_name then
                        addGuard(e,table=node.child.value)
            elseif node.type = trigger_body
                        addAction(e,getExp(node.childs))
        end
        visit_next(node)
    end
```

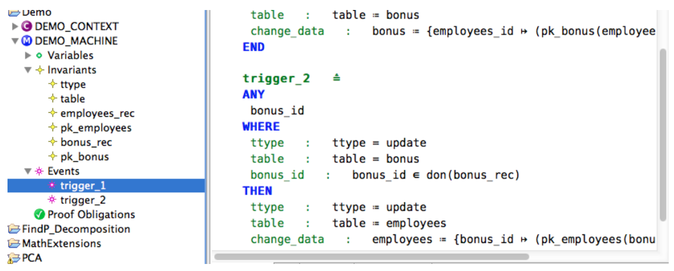
We define a template for using the supporting RODIN API to serialize to Event-B components of the RODIN platform. Figure 3.7(a) shows the

RODIN output files which are generated automatically in the folder *res* by the tool after modeling the scenario described in Section 3.4.1. The output contains full description of two triggers (see Figure 3.7(b)). One thing is missing is the invariant representing the data constraint because we have not defined it in the input SQL file yet. Therefore, we need to add this invariant clause manually to obtain the complete model.



```
▼ META-INF
    MANIFEST.MF
▼ res
  ▼ rodin
      DEMO_CONTEXT.bcc
      DEMO_CONTEXT.bpo
      DEMO_CONTEXT.bpr
      DEMO_CONTEXT.bps
      DEMO_CONTEXT.buc
      DEMO_MACHINE.bum
  ▼ templates
      formula.stg
      string.stg
      xml.stg
  ▼ tests
      test0.sql
```

(a) Rodin project generated



```
Demo
 ▶ DEMO_CONTEXT
 ▼ DEMO_MACHINE
   ▶ Variables
   ▼ Invariants
       ttype
       table
       employees_rec
       pk_employees
       bonus_rec
       pk_bonus
   ▼ Events
       trigger_1
       trigger_2
     Proof Obligations
FindP_Decomposition
MathExtensions
PCA
```

```
 table      :   table ≔ bonus
 change_data    :   bonus ≔ {employees_id ↦ (pk_bonus(employee
END

trigger_2   ≙
ANY
 bonus_id
WHERE
 ttype      :   ttype = update
 table      :   table = bonus
 bonus_id     :   bonus_id ∈ don(bonus_rec)
THEN
 ttype      :   ttype ≔ update
 table      :   table ≔ employees
 change_data    :   employees ≔ {bonus_id ↦ (pk_employees(bonu
```

(b) Output project in RODIN platform

FIGURE 3.7: The modeling result of the scenario generated by Trigger2B

3.6 Chapter conclusions

Database active rules or triggers are one of the most popular application of event-driven architecture in database systems. Modeling and verification of these systems are interesting topic for many research group. Most of their work focused on checking termination property of the trigger execution. While few of them proposed methods for analysing data constraint preservation.

In this chapter, we propose a new method to formalize and verify a database system including triggers with Event-B. In comparison with other existing methods, using Event-B for modeling such kind of systems is well-suited, because ECA rules which are used for describing behaviour of trigger systems are matched to Event-B events. The method proposes a set of translation rules to translate database elements to Event-B constructs. One advantage of the method is that it allows us to prove data constraint preservation properties and detect infinite loops by means of Event-B proof obligations. Another contribution of this chapter is the provided tool which partly supports the automatic translation. It helps to reduce the effort and cost of modeling process. It makes sense because we can bring the formal verification to software development. The initial research result was published in [53] and the extended one with tool is in submission.

Besides the advantages, this method still needs to be improved to model and verify a more complex database systems with more complicated triggers. The current implementation is also limited with only SQLite trigger syntax.

In the next chapter, we will continue to study how to use Event-B and its refinement mechanism to model and verify another important application of event-driven systems, e.g., context-aware systems, which also uses ECA based rules to describe the behavior of the system.

Chapter 4

Modeling and verifying context-aware systems

4.1 Introduction

Context awareness is a computing paradigm that makes applications responsive and adaptive with their environment. Context-aware systems potentially determine their behavior and reduces human-computer interaction by providing knowledge context information of their user's environment. Context awareness of an application relates to adaptation, responsiveness, sensitiveness of the application to changes of the context [42]. In a narrow view, a context-aware system is somehow considered as an event-driven system, i.e., it receives events emitted by context changes and responds to these changes with the providing context knowledge.

Context-aware systems often use context rules to adjust the behavior if the circumstances are changed. It is similar to the structure of database triggers in Chapter 3, these rules can have the form of ECA rules. Furthermore, the behavior of context-ware systems is often complex and uncertain. That could be unacceptable especially when context-aware systems are implemented as safety-critical systems. The results up to date have worked on modeling context awareness with various approaches such as object role modeling, ontology based modeling, logic based modeling [42, 54]. They also have proposed several frameworks for context modeling. However, to the best of our knowledge, there does not exist an approach that models context awareness in several aspects such as events of environments, context rules and uncertainty. Furthermore, the resulted model can be formally verified to ensure the correctness of the system.

In this chapter, we propose to use Event-B as a formal method to model and verify context-aware systems. The contributions of our proposal are: (1) Natural representation of context-aware systems by Event-B concepts. A set of translation rules are proposed to define context awareness components formally. It is a refinement-based method allowing to construct the system gradually (2) After formalization, significant properties are verified via proof obligations of refinement mechanism automatically (or interactively) without any intermediate transformation.

The rest of the chapter is structured as follows: Section 4.2 summarizes some related work. In Section 4.3, we introduce a novel method to model a context-aware system by formalizing its components using Event-B notations. Section 4.4 presents a scenario of an Adaptive Cruise Control system in order to demonstrate our method. We conclude this chapter in Section 4.5.

4.2 Related work

Many papers have been proposed for modeling and verifying context-aware systems with various approaches. Key-value data structure [40, 55] has been used in early works as the simplest method for modeling context awareness. This method exposes many problems since it lacks of interoperability, representation and reasoning mechanism.

Most research efforts that are based on mark-up scheme model have defined and extended markup languages. Henricksen *et al.* [56] proposed to represent contextual data by *Comprehensive Structure Context Profiles* (CSCP). Indulska *et al.* [57] extended CC/CP model to define a set of CC/PP components and attributes to express a various types of context information and context relationships.
Some researchers following the graphical model approach to model contextual data. Mostefaoui [58] presented a three-layered data model for context. Benselim and Seridi-Bouchelaghem [59] recently presented an UML extension for representing and modeling context by creating some stereotypes that are described by several tagged values and some constraints.

Almost all ontology-based approaches have used high-level ontologies to formalize context information and models. Shehzad *et al.* [60] introduced a formal modeling method in context aware systems using OWL. Ejigu *et al.* [61] also proposed ontology based reusable context model that providing structure for contexts, rules and their semantics. The problem with these two pieces of work is that there was no verification mechanism presented.

Besides, Kjaergaard and Bunde-Pedersen [62] proposed a CONAWA calculus that provides mechanism for modeling and interwovenning sets of context-information. However, this approach has some limitations such as probabilistic context information modeling and verification of the system is not discussed yet.

More recently, Tran *et al.* [63] introduced a ROAD4Context framework which is based on Role-Oriented Adaptive Design (ROAD) [64] to model context-aware systems. However, in order to verify the system, it takes more intermediate steps to translate a ROAD4Context model to a Petri net model and then use SPIN to check the system's behaviors. Furthermore, the transformation rules are not presented generally.

In comparison to these works, our method is different as we use Event-B as a modeling method. The advantages of our method are that a context-aware system can be modeled naturally by Event-B notations because the ECA form of context rules is mapped directly to an event. The Event-B refinement allows to develop a context-aware system gradually and ensures the correctness in each refinement stage. After the modeling, the verification process does not require any more translation step. Context constraints are proved mathematically by proving proof obligations which are automatically generated and proved in the supporting tool Rodin.

4.3 Formalizing context awareness

In this section, we consider a simplified context-aware system and represent its components in set theory. Base upon these definitions, we then use Event-B notations to formalize a context-aware system.

4.3.1 Set representation of context awareness

Firstly, we introduce a simple structure of context-aware systems consisting of five components depicted in Figure 4.1. A basic operation of the system is that if there is any change from the environment that can be detected using sensors, it sends events to the core context-aware service. This component then uses both context data entities and context rules to reason about the situation. Finally, it reacts to environment via its behaviors. During that process, the system still has to preserve the constraints.

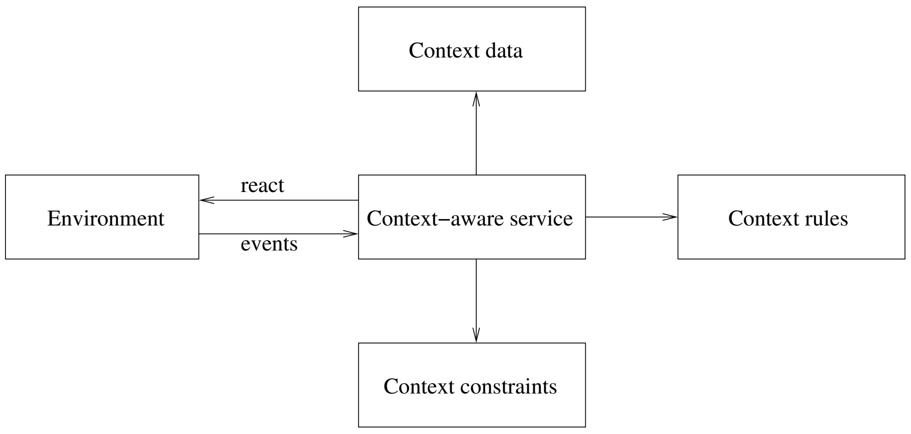


FIGURE 4.1: A simple context-aware system

Definition 4.1 (Context-aware system)**.** A context-aware system is denoted by a 4-tuple, $CaS = \langle E, R, CD, CC \rangle$, where E and R represent for the environment events and context rules respectively, CD denotes context data entities and its relations and CC states the system's constraints.

Definition 4.2 (Environments)**.** Environment is a set of events stated by a set: $E = \mathcal{U}_e$, where e is an event that is sent to context aware core service.

We go further for definitions of context rules and context entities. Let us assume that rules of context-awareness are in the form of ECA (event-condition-action), i.e. if an event e occurs in condition C then do action A. Hence, we present definitions for each element $r, r \in R$ as follows

Definition 4.3 (Context rules)**.** A context rule is used for reasoning and describing the behavior of context-aware systems. It specifies the response actions when a specific event is raised at any condition. Thus, the context rule is denoted by 3-tuple $r = \langle e, a, c\rangle$ where e, c are event and condition of the rule respectively, while a states the action of the rule. Note that, the event e should be included in the event set E.

Context data consists of context entities and their relations. This component takes a role as a data storage of the system.

Definition 4.4 (Context data)**.** Context data is denoted by a 2-tuple $CD = \langle E, R\rangle$, where E is a set of context entities and R is a set of functions mapping between sets of context entities.

4.3.2 Modeling context-aware system

Event-B is based on classical set theory, we thus use it to model context-aware systems according to definitions given in Section 4.3.1. We present transformation rules between a context-aware system and an Event-B model as follows:

- Rule 1: Context data is presented by a set of context entities and their relations. The context entities be treated as a collection of sets and constants. Recall that, an Event-B context consists of set, constant and axioms clauses. The axioms clauses list various predicates of constants in the first order logic formulas. Hence, we can formalize directly a context data by an Event-B context.

- Rule 2: Each event that is emitted by the environment component is represented by an Event-B event. For example: A context-aware system uses a sensor for detecting Wind speed. The sensor regularly detects the environment and this event is sent to the core service. Then, it is then formalized by an event: *detectWind*.

- Rule 3: Since context rule structure has the form of ECA, each rule $r = \langle e, c, a \rangle$ is mapping to an Event-B event. Where e is event that the system senses or received from its environment, c is the additional conditions for reasoning. More precisely, conjunction of e and c are guards of Event-B event while a is mapped to the body of the event (see Table 4.1). All these events are included in either Event-B abstract machines or a refined ones.

Table 4.1: Modeling a context rule by an Event-B Event

IF (e) ON (c)	WHEN $(e \wedge c)$
ACTION (a)	THEN (a) END

- Rule 4: A constraint of the context-aware system is a desired property that the system should maintain. That standpoint matches to the meaning of Event-B invariants, we thus model Context constraints by a set of invariants.

We summarize transformation rules used for modeling in Table 4.2

Table 4.2: Transformation between context-aware systems and Event-B

	Context-aware concepts	**Event-B notations**
Rule 1	Context data CD	Sets, Constants
Rule 2	Context rules $r = \langle e, c, a \rangle$	Events
Rule 3	Environments triggers E	Events
Rule 4	Context constraints CC	Invariants

4.3.3 Incremental modeling using refinement

In fact, the development of context-aware systems often starts from the scratch requirements, then it is built gradually when we have new requirements about context entities and reasoning. For example, more sensors are attached in the system to get various kind of context data. The system also has more context rules to handle with these data. The updated system still has to satisfy context constraints which has been established. Therefore, it requires to have a suitable modeling method for incremental development. As we have described in Section 4.3.2, a context-aware system is translated to an abstract Event-B model. It is apparently suitable for modeling the initial stage of a context-aware system. In this subsection, we answer the question how our approach fits to incremental development of such systems.

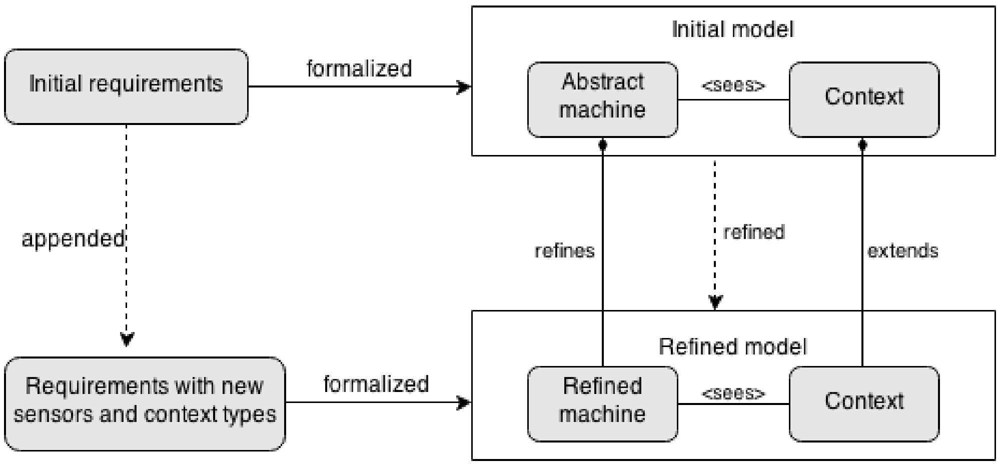


FIGURE 4.2: Incremental modeling using refinement

The refinement mechanism of Event-B makes it possible to model context-aware systems incrementally. We already know that Event-B provides both superposition refinement and vertical refinement. In the former, the abstract variables are retained in the concrete machine, with possibly some additional variables, hence it is suitable for modeling a context-aware system which is often extended by adding new sensors. The proposed incremental modeling method is illustrated in Figure 4.2.

- Static part: For example, when a new sensor is added to the system, we may have to deal with new context data types. The context data can be an extension of the old one. Applying Rule 1, we formalize it as a new Event-B context which extends the ones in abstract model.
- Dynamic part: We begin with abstract machines to model the general behavior of the very beginning system, after that we refine these machines by concrete ones to represent new requirements of the systems. For instance, adding new sensors usually generate new appropriate events. Hence, the system also needs more rule to react to these events. In the refined machines, new added variables can refer to new context data elements. The events of a new refined machine can refine the abstract ones to describe the system more precisely. With the dynamic part, we need to prove that a new model in the increment step needs to satisfy the context constraints including ones defined in the early stages. That proof can be achieved by using INV proof obligations which states that invariant preservation PO can be checked at any refinement. According to Rule 4 in Section 4.3.2, all constraints are represented by Invariants, therefore a new constructed context-aware system at any refined step preserves all constraints of the initial step.

4.4 A case study: Adaptive Cruise Control system

We demonstrate our approach by modeling a scenario of an Adaptive Cruise Control (ACC) system. First, we introduce the scenario, then we apply the modeling method presented in Section 4.3.2 and then we verify context constraint preservation properties of the system.

4.4.1 Initial description

ACC controls car's speed is based on the driving conditions which are enhanced with a context-aware feature such as target detection. The ACC system use a sensor to detect target in front of the car. The car has a maximum speed which is initially set to a value. If the car does not detect a target then ACC increase the speed, other wise decreases the speed with constant amount. If the car is stopped and there is no target detected then it is resumed with initial speed.

The ACC must conform to a context constraint such that the speed is always in safe range, i.e., the speed is less or equal to the maximum speed.

4.4.2 Modeling ACC system

In this scenario, there are three sensors, following the approach presented in Section 4.3, we specify the initial system with one abstract machine and one context, namely ACC_M0 and $Target$.

- Applying Rule 1, context $Target$ represents context data received from the target detection sensor. More precisely, the information sent by the sensor let us know whether there is an object which is currently in front of the car. We thus formalize the context entities as a set $TARGET\_DETECTION$ which is equivalent to $BOOL$ set.
- The sensor will periodically send context data to the system by emitting events. We need to check if context data contains the

information of obstacle objects. Hence, we specify two Event-B events *TargetDetected*, *TargetUndetected* correspondingly.

- Context rules are used for specifying the system behavior which depends on the context. We attract three rules from the initial description and translate them to three corresponding events following Rule 3.
- Applying Rule 4, the context constraint is translated to invariant CXT_CSTR.

Moreover, a variable *speed* in the model specifies the speed of the car. The Event-B specification of the ACC system is partially illustrated in Figure 4.3.

Strengthen guards: With the initial description, the generated INV POs of the translated Event-B model are failed to prove. More precisely, with event *TargetDetected*, PO $TargetDetected/CXT\_CSTR/INV$ is generated as follows:

$$speed \leq MAX\_SPEED \wedge target = FALSE \vdash speed + INC\_SPEED \leq MAX\_SPEED \quad (4.1)$$

Sequent 4.1 is failed if $MAX\_SPEED+INC\_SPEED < speed \leq MAX\_SPEED$ before event execution.

To make the model more precise, we strengthen the context rules by adding more conditions. i.e. the event guards have more clauses (Figure 4.4)

```
CONTEXT  Target
CONSTANTS
      TARGET_DETECTION
      MAX_SPEED
      INC
AXIOMS
      axm1 : TARGET_DETECTION  =
             BOOL
      axm2 : MAX_SPEED ∈ ℕ
      axm3 : INC < MAX_SPEED
      axm4 : INC ∈ ℕ
END
```

```
MACHINE  ACC_M0
SEES  Target
VARIABLES
      speed
      target_det
INVARIANTS
      inv1 : speed ∈ ℕ
      inv2 : target_det ∈ TARGET_DETECTION
      inv3 : speed ≤ MAX_SPEED
EVENTS
Initialisation
      begin
            act1 : speed := MAX_SPEED
      end
Event  TargetDetected ≙
      when
            grd1 : target_det = TRUE
      then
            act1 : speed := speed − INC
      end
Event  TargetUndetected ≙
      when
            grd1 : target_det = FALSE
      then
            act1 : speed := speed + INC
      end
END
```

FIGURE 4.3: Abstract Event-B model for ACC system

4.4.3 Refinement: Adding weather and road sensors

At this stage, the system has more sensors to be aware more precisely of the current context situations. Weather and road sensors are attached to the system. Similarly to target detection sensor, they send the context data periodically to the system. Context rules of the system are also extended for reacting to new added sensors as follows: "When a car travels in a raining condition or sharp bend, ACC reduces car's speed". With new sensors, the system need to fulfil the constraint such as "The

```
EVENTS
Event   TargetDetected ≙
     when
          grd1 : target_det = TRUE
          grd2 : speed > INC
     then
          act1 : speed := speed − INC
     end
Event   TargetUndetected ≙
     when
          grd1 : target_det = FALSE
          grd2 : speed < MAX_SPEED − INC
     then
          act1 : speed := speed + INC
     end
END
```

FIGURE 4.4: Events with strengthened guards

speed can not be equal to maximal speed if it is raining or the road is sharp".

Refined model: Following the method presented in Section 4.3.3, context *Weather_Road* extending context *Target* represents context data of two new sensors. We use two constant sets *RAINING* and *SHARP* to represent context data from sensors. Machine ACC_M1 is the concrete machine specifying the behavior of the system updated with new requirements.

Since two new rules are appended to the requirements, two corresponding events are added for this machine. The first one representing a new added rule is not extended. This event *RainSharp* describes the behavior of the system when sensors send data indicating that it is raining or the road is sharp. While the second one *TargetUndetected* refines event of the abstract model. The context constraint is formalized as an invariant *cxt_ct*. The Event-B specification of the refined model is partially illustrated in Figure 4.5.

```
MACHINE ACC_M1
REFINES ACC_M0
SEES Weather_Road
VARIABLES
      isRain
      speed
      target_det
      isSharp
INVARIANTS
      inv1 : isRain ∈ RAINING
      cxt_ct : isRain = TRUE ∨ isSharp =
           TRUE ⇒ speed < MAX_SPEED
      inv3 : isSharp ∈ SHARP
EVENTS
Initialisation
      begin
           skip
      end
Event TargetUndetected ≙
extends TargetUndetected
      when
           grd1 : target_det = FALSE
           grd2 : speed < MAX_SPEED − INC
           grd3 : isRain = FALSE
           grd4 : isSharp = FALSE
      then
           act1 : speed := speed + INC
      end
Event RainSharp ≙
      when
           grd1 : isRain = TRUE ∨ isSharp =
                TRUE
      then
           act1 : speed := speed − INC
      end
END
```

```
CONTEXT Weather_Road
EXTENDS Target
CONSTANTS
      RAINING
      SHARP
AXIOMS
      axm1 : RAINING = BOOL
      axm2 : SHARP = BOOL
END
```

FIGURE 4.5: Refined Event-B model for ACC system

4.4.4 Verifying the system's properties

The system should always satisfy context constraint preservation properties. It means that the context constraint is preserved before and after using the context rules to adapt context changes. With proposed method, context constraints are translated to invariant clauses. Consequently, we prove the system's correctness by proving proof obligations of such invariants.

TABLE 4.3: Proof of context constraint preservation

$target\_det = TRUE \Rightarrow speed < MAX\_SPEED$ $target\_det = TRUE$ $speed > INC$ $\vdash$ $target\_det = TRUE \Rightarrow speed - INC < MAX\_SPEED$	TargetDetected/ctx_ct1/INV

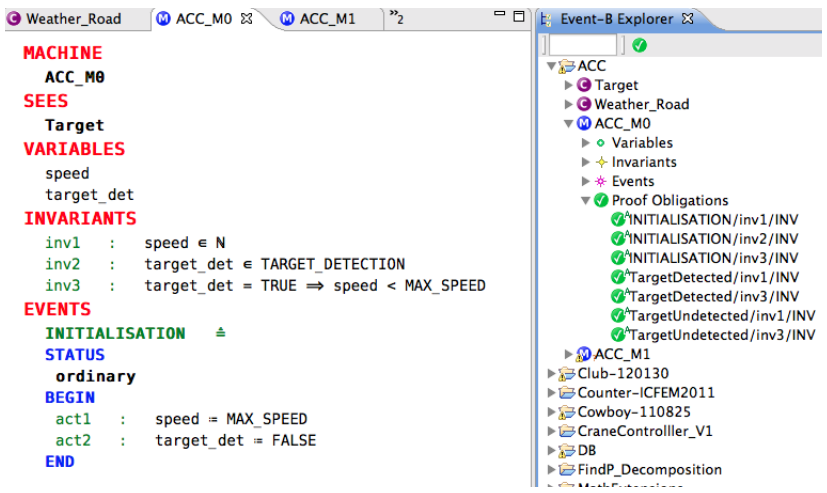
The proof obligations (PO) for these invariants of both abstract and refined machines as follows:

- Machine ACC_M0: "*TargetDetected/ctx_ct1/INV*" (Figure 4.3) and "*TargetUndetected/ctx_ct1/INV*"
- Machine ACC_M1: "*TargetUndetected/ctx_ct/INV*" and "*RainSharp/ctx_ct/INV*"

These POs are generated and proved automatically with the Rodin tool as illustrated in Figure 4.6. Hence, the ACC system always satisfies predefined context rules.

4.5 Chapter conclusions

The use of context-awareness plays an important role in reactive and interactive systems. Context-aware systems which follows the event-driven


Weather_Road | ACC_M0 | ACC_M1 | »2

```
MACHINE
  ACC_M0
SEES
  Target
VARIABLES
  speed
  target_det
INVARIANTS
  inv1   :   speed ∈ ℕ
  inv2   :   target_det ∈ TARGET_DETECTION
  inv3   :   target_det = TRUE ⇒ speed < MAX_SPEED
EVENTS
  INITIALISATION   ≙
  STATUS
   ordinary
  BEGIN
   act1   :   speed ≔ MAX_SPEED
   act2   :   target_det ≔ FALSE
  END
```

Event-B Explorer

```
▼ ACC
  ▶ Target
  ▶ Weather_Road
  ▼ ACC_M0
    ▶ Variables
    ▶ Invariants
    ▶ Events
    ▼ Proof Obligations
        INITIALISATION/inv1/INV
        INITIALISATION/inv2/INV
        INITIALISATION/inv3/INV
        TargetDetected/inv1/INV
        TargetDetected/inv3/INV
        TargetUndetected/inv1/INV
        TargetUndetected/inv3/INV
  ▶ ACC_M1
▶ Club-120130
▶ Counter-ICFEM2011
▶ Cowboy-110825
▶ CraneControlller_V1
▶ DB
▶ FindP_Decomposition
```

FIGURE 4.6: Checking properties in Rodin

architecture are applied in many real-world systems such as mobile, embedded systems, etc..The behavior of such systems can be expressed by context-rules which has the form of ECA. Modeling and verifying context-aware systems are difficult tasks due to their complex behavior. In this chapter, we introduce a new method to model and verify such systems. The advantages of our method over the existing methods and approaches are natural representation of context-aware concepts to model and the use of invariant preservation proof obligations generated by refinement mechanism in Event-B to verify the correctness of the system. Refinement of Event-B machines makes our proposed method suitable for incremental modeling of such systems. The content of this chapter was published in [65, 66].

We, however, just consider a simple case of context awareness in this chapter. Context data is treated as boolean sets. Since, we use invariant clauses to specify desired properties, it is limited to safety properties. The other properties such as liveness can not be expressed directly by invariants. Furthermore, one of important characteristic of context-aware systems is uncertain. In this chapter, however, we just consider that the system are described by precise requirements with exact numerical data (for example: we subtract a constant integer value from the current speed, context data values are either TRUE or FALSE).

In Chapter 3 and Chapter 4, we propose new methods for modeling and verifying two particular types of event-driven systems. In both systems, the behavior can be represented by ECA structure and is described by precise requirements. In the next chapter, we will consider the case that a system which is described by uncertain or vague requirements. We will propose a novel modeling and verification method for such systems using Event-B.

Chapter 5

Modeling and verifying imprecise system requirements

5.1 Introduction

Formal methods are mathematical techniques for describing system model properties. Such methods providing frameworks to specify and verify the correctness of systems as well as event-driven ones requiring precise description. In Chapter 4, we model context-aware systems where they are described by precise descriptions. However, we often are faced with imprecise descriptions where ambiguous, vague or uncertain terms such as “very cold”, “far”, or “low important”, are used because the stakeholders usually do not care much about describing the system precisely. Therefore, frameworks which are formal enough to be used for analysing as well as representing imprecise requirements are desirable.

The method with the Fuzzy set, proposed by Zadeh [28], is one such formal framework, where the Fuzzy If-Then rules are sometimes employed to represent imprecise system requirements. Informal statements expressed in natural languages such as “very far” or “too close” can be well encoded in the Fuzzy set, which enables further analysis on the specifications. It involves continuous numerical reasoning since the Fuzzy set is essentially based on the idea of representing the fuzziness degree in terms of Real numbers between 0 and 1.

In general, system requirements include functional specifications, whose various properties are checked at this same level of abstractions before starting further development steps. The requirements written in terms of Fuzzy If-Then rules can be an adequate representation, but require further techniques for checking properties formally, which may elucidate perspectives different from those for detecting and resolving conflicts. The Fuzzy If-then rules are translated into other formal frameworks such as PetriNet [67, 68] or Z notation [69, 70].

Common properties of software systems as well as event-driven systems are safety and liveness properties. While safety properties guaranty that bad things do not happen, liveness properties (e.g. termination, eventuality, progress, persistence) assure that the system will reach a defined good state. In this chapter, we present a new refinement-based method to model and verify both safety and eventuality properties of the system. In particular, we apply the proof methods proposed in [71] to verify the eventuality properties. As far as we know, this thesis reports the first concrete results of formal checking of such properties for imprecise system requirements.

This chapter employs Event-B refinement to model event-driven systems which are described by a set of Fuzzy If-Then rules. The contributions of this chapter are as follows: (1) providing a set of translation rules from the Fuzzy If-then rules to Event-B language constructs (2) making use of Event-B refinement to formalize timed Fuzzy If-Then rules.

(3) providing a set of translation rules to formalize eventualities, which makes use of the refinement modeling approach that Event-B supports, (4) demonstrating how both safety and eventuality properties of a set of the Fuzzy If-Then rules are verified with RODIN/Event-B.

The rest of this chapter is structured as follows. Section 5.2 reviews related work. In Section 5.3, we present representation of fuzzy terms in classical sets. Using such representation, we first introduce a set of translation rules to model fuzzy If-Then rules. We then present timed Fuzzy If-Then rules to formalize timed systems. We then show how to model safety properties and eventuality properties of imprecise requirements in Event-B. Section 5.5 presents an example of Crane controller to illustrate the proposed method in detail. Conclusions are given in Section 5.6.

5.2 Related work

Several approaches have been addressed to modeling and verification of fuzzy requirements using formal methods, however, these are different from our method. C.Matthews and Paul A. Swatman introduced a fuzzy logic toolkit for the formal specification language Z [69, 70]. This toolkit defines the operators, measure and modifiers necessary for the manipulation of fuzzy sets and relations. A series of laws are provided that establish an isomorphism between conventional Z and the extended notation when applied to boolean sets and relation. It can be modeled as a partial rather than total function. The focus is on the specifications of the rule base and the operations necessary for fuzzy inferences. However, they do not incorporate the notion of refinements. It just provides definition and manipulation of fuzzy sets and relations by using Z.

V.Pavliska and J.Knybel [72, 73] introduced modified Petri Nets as a tool for fuzzy modeling. Basic concepts and relations between Fuzzy Petri Nets and Fuzzy IF-THEN rules are described and an algorithm for decomposition of fuzzy Petri net into set of linguistic descriptions are presented and its implementation mentioned. Their work just showed how to model the system and does not mention how to verify the system properties.

B.Intrigila *et al.* [67] has introduced a verification method of fuzzy control systems using model-checking technique with Murphi verifier. The authors eased the modeling phase by using finite precision real numbers and external C functions.

S.J.H.Yang *et al.* [68] proposed to use high-level Petri Net in order to verify fuzzy rule-based systems. This method can detect the system's errors such as redundancy, inconsistency, incompleteness, and circularity but it has to take extra step to normalize the rules into Horn clauses before transforming these rules to and use incidence matrix as fixed-value matrix for degree membership.

J.Lee *et al.* [74] extended object oriented modeling by using fuzzy objects to capture and analyze imprecise requirements. Even this approach is straight to show how the fuzzy classes can be mapped into XML schemas and XML documents [75], we can not verify these requirements with the fuzzy object model.

M.Goncalves *et al.* [76] presented an approach to a database application method which translates the formal specifications to implementations in the structured query language (SQL) enhanced with fuzzy logic (SQLf). This method allows to extend the tuple calculus in order to express fuzzy queries.

5.3 Modeling fuzzy requirements

In this section, we first present an approach to representing fuzzy terms by traditional set theory. Based upon these representation, we introduce translation rules from set of Fuzzy IF-Then rules to Event-B. Then we make use of Event-B refinement to model timed systems.

5.3.1 Representation of fuzzy terms in classical sets

As stated in Chapter 2, we may use fuzzy sets and fuzzy logic to formalize imprecise system requirements. There are several different classes of hedge operations, each represented by a linguistic construct. There are hedges that intensify the characteristic of a fuzzy set (very, extremely), dilute the membership curve (somewhat, rather, quite), form the complement of a set (not), and those that approximate a fuzzy region or convert a scalar to a fuzzy set (about, near, close to, approximately). Hedges play the same role in fuzzy production rules that adjectives and adverbs play in English sentences. The application of a hedge changes the behavior of the fuzzy sets, in the same way that adjectives change the meaning and intention of an English sentence. Fuzzy logic usually uses IF-THEN rules, or equivalent constructs, such as fuzzy associative matrices. Rules are usually expressed in the form: IF $\langle \mathit{variable} \rangle$ be $\langle \mathit{property} \rangle$ THEN $\langle \mathit{action} \rangle$.

Example: IF costs are very high THEN margins are small.

We will show that imprecise requirements using fuzzy sets and fuzzy rules can be represented by classical sets. First, the general form FR, also called well-defined form, of a imprecise requirement can be represented as:

$$\text{IF } x \text{ is } \delta Y \text{ THEN } m \text{ is } \gamma P$$

Recall that, in classical set theory, sets can be combined in a number of different ways to produce another set such as Union, Intersection, Difference, Cartesian product. Below we recall some definitions related to Cartesian product operation, the definition of an ordered pair and Cartesian product of two sets using it. Then the Cartesian product of multiple sets is also defined using the concept of n-tuple.

Definition 5.1 (ordered pair)**.** An ordered pair is a pair of objects with an order associated with them. If objects are represented by x and y, then we write the ordered pair as $\langle x, y\rangle$.

Definition 5.2 (Cartesian product)**.** The set of all ordered pairs $\langle a, b\rangle$ where a is an element of A and b is an element of B, is called the Cartesian product of A and B and is denoted by $A \times B$.

Example 1: Let A = {1, 2, 3} and B = {a, b}.

Then $A \times B = \{\langle 1, a\rangle, \langle 1, b\rangle, \langle 2, a\rangle, \langle 2, b\rangle, \langle 3, a\rangle, \langle 3, b\rangle\}$.

The concept of Cartesian product can be extended to that of more than two sets. First we are going to define the concept of ordered n-tuple.

Definition 5.3 (ordered n-tuple)**.** An ordered n-tuple is a set of n objects with an order associated with them (rigorous definition to be filled in). If n objects are represented by x_1, x_2, ..., x_n, then we write the ordered n-tuple as $\langle x_1, x_2, ..., x_n\rangle$.

Definition 5.4 (Cartesian product)**.** Let A_1, ..., A_n be n sets. Then the set of all ordered n-tuples $\langle x_1, ..., x_n\rangle$ where $x_i \in A_i$, $\forall\ i = \overline{1, n}$, is called the Cartesian product of $A_1, ..., A_n$, and is denoted by $A_1 \times ... \times A_n$.

Corollary 5.5. *A collection of well-defined fuzzy requirements can be specified by classical sets.*

Proof. Suppose that, fuzzy requirements of a system are specified by $FR = \{FR_i\}$, $FR_i = \{x_i, m_i, \delta_i, \gamma_i, Y_i, P_i\}$, $i = \overline{1, n}$. Clearly that, x_i, m_i are considered as variables in the specification, P_i is obviously described by a set of values. We consider if $\delta_i Y_i$ can be specified by a classical set in which δ_i is a hedge, Y_i is a generator of a fuzzy clause. As FR is a finite set of rules so hedges and generators in the system can be established two different finite sets, δ and Y, respectively. According to the Definition 5.2, $\delta_i Y_i$ and $\gamma_i P_i$ are a membership of the Cartesian product of two sets $\delta \times Y$ and $\gamma \times P$ respectively. Consequently, every elements in FR can be specified by classical sets. ■

5.3.2 Modeling discrete states

We start to model the discrete behavior of the system which is described by imprecise requirements. Then, we inherits the approach introduced by J.Abrial *et al.* [77] to model the continuous behavior.

We will explain how imprecise requirements in Fuzzy If-then rules are modeled using discrete states. Suppose that, a system is specified by a collection of requirements FR_i :

$$\textbf{if } x_i \text{ is } \delta_i Y_i \textbf{ then } m_i \text{ is } \gamma_i P_i$$

According to the Corollary 5.5, the above requirements can be represented by classical sets. Since Event-B is a language based on the classical theory set, we propose an approach to modeling the system with Event-B method. A system consisting a collection of requirements $FR_i, i = \overline{1, n}$ is modeled by an Event-B model $FR_B = \langle FR\_C, FR\_M \rangle$, where FR_C and FR_M are Event-B context and machine respectively.

We propose below partial transformation rules to map fuzzy requirements to Event-B's elements. The important principle of the transformation process is that we can preserve the structure and represent all fuzzy requirements using the Event-B notation. Moreover, safety properties must be preserved by actions of the system.

Transformation rules:

- Rule 1. All hedges δ_i and γ_i, generators Y_i and values P_i in the collection of requirements are translated to three sets δ, γ, Y, and P respectively. They are stated in the SETS clause of FR_C.
- Rule 2. Linguistic variables x_i, m_i in each FR_i are mapped to variables x_i, m_i of the Event-B machine FR_M.
- Rule 3. Each variable x_i is described as a membership of a Cartesian product of two sets $\delta \times Y$, m_i is described as a membership of a Cartesian product of two sets $\gamma \times P$ (Corollary 5.5).
- Rule 4. Each requirement FR_i is modeled by an event ev_i in Event-B machine FR_M.

Note that, these are only partial transformation rules, we need to give more additional parts to obtain the completed Event-B specification (Figure 5.1).

5.3.3 Modeling continuous behavior

Currently we use Fuzzy If-Then rules just to formalize time independent systems. In fact, most of real world systems depend on time. In this section, we extend the method proposed in Section 5.3.2 with timed fuzzy if-then rules. We also make use of Event-B refinement to model

CONTEXT *FR_C*
SETS
  δ
  γ
  P
  Y
END

MACHINE *FR_M0*
SEES *FR_C*
VARIABLES
  x_i
  m_i
INVARIANTS
  inv1 : $\mathtt{x_i} \in \mathbb{P}(\delta \times \mathtt{Y})$
  inv2 : $\mathtt{m_i} \in \mathbb{P}(\gamma \times \mathtt{P})$
  inv3 : $\mathcal{I}$
EVENTS
Event $FR_i \mathrel{\widehat{=}}$
  when
    grd1 : $\mathtt{x_i} = \{\delta_\mathtt{i} \mapsto \mathtt{Y_i}\}$
  then
    act1 : $\mathtt{m_i} := \{\gamma_\mathtt{i} \mapsto \mathtt{P_i}\}$
    act2 : $\mathtt{x_i} := \{\delta_\mathtt{j} \mapsto \mathtt{Y_j}\}$
  end
END

FIGURE 5.1: A part of Event-B specification for discrete transitions modeling

timed systems. Then we will be able to check significant time dependent properties of these systems. First, we define timed Fuzzy If-Then rules that has the form as follows:

$$\text{IF x(t) is A THEN y(t) is B}$$

This form is similar to the one defined in fuzzy set except that linguistic variables are now dependent on variable t representing for time clock.

Based on this observation, we refine the abstract model in several steps to get more precise model. Inspired by the approach presented in [77], if a fuzzy requirement *FR_i* contains any time-dependent variable, then we will refine the appropriated event of the abstract machine. Also, we have introduced in Rule 4 that a variable x is specified as one member of a Cartesian product of two sets $\delta \times Y$, where δ is a set of fuzzy hedges, Y is a fuzzy set. Similarly, we add a new continuous variable x_c of the refined machine which is formalized by a time dependent function such as $x_c \in \mathbb{R} \nrightarrow \delta \times P$.

MACHINE FR_M1
REFINES FR_M0
VARIABLES
    x_c
    t
    m
INVARIANTS
    inv1 : $\mathtt{x_c} \in \mathbb{R} \rightarrow \mathbb{P}(\delta \times \mathtt{Y})$
    inv2 : $\mathtt{m} \in \delta \times \mathtt{P}$
    inv3 : $\mathtt{t} \in \mathbb{R}_{\geq 0}$
    inv4 : $\mathtt{x_c}(\mathtt{t}) = \mathtt{x_i}$
END

FIGURE 5.2: A part of Event-B specification for continuous transitions modeling

We introduce two rules for modeling continuous transitions the requirements as follows:

- Rule 5: If either variable x_i or m_i (in a fuzzy requirement FR_i) depends on time, then its corresponding event will be refined. A new variable $t.t \in \mathbb{R}$ for representing time clock is added to the refined machine.
- Rule 6: The time dependent variables (in Rule 1) are replaced by time functions and glue invariants in the refined machine.

The abstract machine FR_M is refined by a concrete machine FR_M_C illustrated in Figure 5.4. A new variable t represents for time clock. We also add a new clause $inv4$ which is glue invariant of refined and abstract variable, i.e. x_c and x_i respectively.

5.4 Verifying safety and eventuality properties

5.4.1 Convergence in Event-B

Convergence property of an Event-B machine is the convergence of a set of its events. It means that a set of events can not run forever. As a consequence, the other events eventually happen. These events are called convergent events. To prove this property, Event-B provides a mechanism to use an variant V which maps to state variable v to a Natural number, then these events are proved to decrease the variable v. More specifically, let e be a convergent event, v and v' are state before/after executing e, then we prove that $V(v') < V(v)$. Two proof obligation rules are generated for every convergent event where VAR (5.1) ensures the decrement of the variant and NAT (5.2) makes sure that the variant is a natural number after the event execution:

$$I(v), G(x, v), Q(x, v, v') \vdash V(v') < V(v) \tag{5.1}$$

$$I(v), G(x, v) \vdash V(v) \in N \tag{5.2}$$

Besides mapping a variant V to a Natural number, V can refer to a finite set. In this case, we prove VAR proof obligation by showing that $V(v') \subset V(v)$. More precisely, it is illustrated in Sequent 5.3.

$$I(v), G(x, v), Q(x, v, v') \vdash V(v') < V(v) \tag{5.3}$$

The NAT PO is now not generated, instead we just need to prove $V(v)$ is a finite set once by using FIN proof obligation (5.4). This proof obligation does not depend on a set of convergent and anticipated events.

$$I(v) \vdash \mathit{finite}(V(v)) \tag{5.4}$$

We also recall some properties of an Event-B machine such as convergence and deadlock-free as follows:

- Machine M is convergent in a state formula P if any trace of its execution does not end with an infinite sequence of P-states.
- Machine M is Convergent in state formula P if the infinite trace of M ends with an infinite sequence of P-states.
- Machine M is deadlock-free in P if it does not end in a P-state.

5.4.2 Safety and eventuality analysis in Event-B

Event-B provides the way to express safety properties directly by using the invariants. Hence, we can prove the correctness of these properties using INV proof obligation. Recall that, the INV proof obligation ensure that a invariant is maintained after and before executing an event. It is formally defined as

$$\mathcal{I}(v) \wedge A(v) \wedge G(v) \wedge S(v, v') \vdash \mathcal{I}(v')$$

where v is the variable, v' is the new value after assignment S, A is a set of axioms, G is the event guard.

Event-B does not support to specify liveness properties directly but we can follow the approach [71] to verify properties such as *existence* ($\Box\Diamond P$), *progress* ($\Box(P_1 \Rightarrow \Diamond P_2)$), *persistence* ($\Diamond\Box P$), where P is any first order logic formula, $\Diamond$ and $\Box$ are standard operators of Linear Temporal Logic

(LTL), under weakness assumption. We will discuss here in detail *existence* property.

Let M be a machine and P is a first-order logic formula. Assume that a given machine M with n events $e_i (i \in \overline{1..n})$, $e_i =$ any x where $G(x, v)$ then $A(x, v, v')$ end. They claim that if M is convergent in $\neg P$ and M is deadlock-free in $\neg P$ then $\Box\Diamond P$ is satisfied in M. This approach uses the variant clause to prove convergence of a machine and we introduce an auxiliary refined machine at the last refinement process to apply this proof method. The method is illustrated as the following steps:

1. Define an integer variant $V(v)$
2. Check if machine M is deadlock-free in P
3. For each event e_i
   - Check if e_i is convergent in P
4. if M is convergent and deadlock-free in P, then M will eventually satisfies P.

5.4.3 Verifying safety properties

Generally, safety properties are expressed directly by the invariants. Hence, safety properties of a systems described by a collection of Fuzzy If-Then are translated directly to invariants. We will show that the translation assure preservation of safety properties through imprecise requirements.

Corollary 5.6. *With the modeling proposed in transformation rules, the safety properties are preserved by all actions in imprecise requirements of the system. It is formally define as* $M \vdash I \Rightarrow FR \vdash S$

Proof. Suppose that, a collection of imprecise requirements $FR = \{FR_i\}$, $i = \overline{1, n}$, is specified by corresponding event evt_i. Safety properties of the system are specified in the invariant $\mathcal{I}$. We have to prove that safety constraints are preserved through all requirements by showing that it remains true before and after firing (executing) each event. This is obviously achieved through proof obligations of the Event-B machines which is used to preserve their invariants.

Without loss of generality, we assume that the imprecise requirements and constraints contain one variable v, hence we need to prove:

$$\mathcal{I}(v) \wedge evt_i(v, v') \vdash \mathcal{I}(v')$$

This predicate allows us to ensure the safety properties after executing the events in model, which is exactly the form of a proof obligation generated from Event-B machine and we can analyse them using the support tool (Rodin). Therefore, the safety properties stated in requirements are shown preserved. ■

5.4.4 Verifying eventuality properties

Recall that, the paper [71] introduced reasoning techniques to prove classes of liveness properties such as *existence*, *progress*, *persistence*. They claims that with a state formula P which is a first-order logic formula and an Event-B machine M that is convergent and deadlock-free in P then $\neg P$ will *always eventually* ($\Box\Diamond\neg P$) holds.

In order to reason about eventuality properties on a set of fuzzy If-Then rules, we first map fuzzy values to Natural numbers. Since fuzzy sets can be represented by classical sets consisting of discrete values (Section 5.3.1), the mapping on Natural numbers instead of a continuous range

[0..1] is acceptable. Therefore, we give a new definition of fuzzy sets as follows

Definition 5.7 (Fuzzy set)**.** A fuzzy set is a pair $\langle U, \mu \rangle$, where U is a set and μ is the membership degree function, can be represented as a pair $\langle P, \mu_s \rangle$, where P is a crisp set, μ_s is a total function such that $\mu_s : P \to \mathbb{N}$

Similarly, we also use a total function $\mu_h : \delta \to \mathbb{N}$ as mapping values of fuzzy hedges.

We already state that a system is specified by a collection of requirements FR_i :

$$\textbf{if } x_i \text{ is } \delta_i Y_i \textbf{ then } m_i \text{ is } \gamma_i P_i$$

We propose a refinement-based approach to modeling with an introduction of additional translation rules to extend the context and refine the machine of the abstract model as follows

- Rule 7. Fuzzy values of each element in P, Y and hedges δ are translated to total functions $deg_P : P \to \mathbb{N}$, $deg_y Y : Y \to \mathbb{N}$, $deg_H : \delta \to \mathbb{N}$ respectively.
- Rule 8: Adds a variant mapping to linguistic variable that appears in eventuality property expression Q.
- Rule 9. Refines each event representing for fuzzy if-then requirements by two events: a convergent and an ordinary one.
- Rule 10. Adds a clause $\neg Q(x_i)$ to the guards of each convergent event, and a clause $Q(x_i)$ to the ordinary one.

- Rule 11. Deadlock free property is encoded as a theorem of the refined machine.

A partial Event-B specification for these rules is depicted in Figure 5.3.

CONTEXT *FR_C2*
EXTENDS *FR_C*
CONSTANTS
 deg_P
 deg_Y
 deg_δ
AXIOMS
 $deg_P \in P \rightarrow \mathbb{N}$
 $deg_Y \in Y \rightarrow \mathbb{N}$
 $deg_H \in \delta \rightarrow \mathbb{N}$
END

MACHINE *FR_M2*
SEES *FR_C1*
REFINES *FR_M1*
VARIABLES
 vx
INVARIANTS
 inv4 : $\mathtt{vx} \in \mathbb{N}$
VARIANT
 vx
EVENTS
Event $ev_i\_CE \mathrel{\widehat{=}}$
Status convergent
extends evt_i
 when
  grd1 : $\mathtt{x_i} = \{\delta_\mathtt{i} \mapsto \mathtt{Y_i}\}$
  grd2 : $\neg\, \mathtt{Q}(\mathtt{x_i})$
  grd3 : $\mathtt{vx} := \mathtt{deg_Y}(\mathtt{Y_i})$
 then
  act1 : $\mathtt{m_i} := \{\delta_\mathtt{i} \mapsto \mathtt{P_i}\}$
  act2 : $\mathtt{x_i} := \{\delta_\mathtt{j} \mapsto \mathtt{Y_j}\}$
  act3 : $\mathtt{vx} := \mathtt{deg_Y}(\mathtt{Y_j})$
 end
Event $ev_i\_OE \mathrel{\widehat{=}}$
Status ordinary
extends evt_i
 when
  grd1 : $\mathtt{x_i} = \{\delta_\mathtt{i} \mapsto \mathtt{Y_i}\}$
  grd2 : $\mathtt{Q}(\mathtt{x_i})$
  grd3 : $\mathtt{vx} := \mathtt{deg_Y}(\mathtt{Y_i})$
 then
  act1 : $\mathtt{m_i} := \{\gamma_\mathtt{i} \mapsto \mathtt{P_i}\}$
  act2 : $\mathtt{x_i} := \{\delta_\mathtt{j} \mapsto \mathtt{Y_j}\}$
  act3 : $\mathtt{vx} := \mathtt{deg_Y}(\mathtt{Y_j})$
 end
END

FIGURE 5.3: A part of Event-B specification for eventuality property modeling

Before showing that if a collection of requirements satisfy a eventuality property $Q(x)$, we introduce definitions relating to some properties of fuzzy rules.

Definition 5.8 (Convergence)**.** Fuzzy rules are convergent from a state $Q(x)$ if each rule decreases value of variable x. It is formally defined as: $FR_i, Q(x) \vdash x' < x$, where x' is value after executing rule FR_i.

Definition 5.9 (Deadlock-freeness)**.** Fuzzy rules are deadlock-free in a state $Q(x)$ if IF clause of at least one rule is satisfied. It is formally defined as

$$Q(x) \Rightarrow \bigvee_{i=1}^{n} (\exists\, x_i . x_i = \delta\, Y_i)$$

Corollary 5.10. *With proposed translation rules, if a collection of Fuzzy If-Then rules* $\{FR\}$ *are convergent and deadlock-free from a first-order logic state formula* $Q(x)$ *where* x *is a linguistic variable then the state property* $\neg Q(x)$ *will always eventually holds. Formally, we have* $\{FR\} \vdash \Box\Diamond\neg Q(x)$.

Proof.

Suppose that, a collection of fuzzy if-then rules $FR = \{FR_i\}$, $i = \overline{1,n}$, is first formalized by an abstract machine FR_M1, then is refined by another machine FR_M2 containing a set of convergent events evt_i.

Applying Rule 9, each fuzzy rule is represented by a convergent event evt_i with guard $G(x)$. Following Rule 10, a new clause $\neg Q(x)$ is added to the guard condition of each convergent event.

According to the translation Rule 6 and 7, approximation of a linguistic variable x is a natural number and is mapped to an variant $V(x)$. Furthermore, each fuzzy rule decreases the fuzzy variable x (Definition 5.8), i.e $V(x') < V(x)$. Hence, we have

$$\mathcal{I}, G(x), \neg Q(x) \vdash V(x') < V(x).$$

This predicate is the form of VAR proof obligation generated from Event-B machine to prove that all events of the machine FR_M2 are convergent (*).

We already state that fuzzy rules are deadlock free in $Q(x)$, according to Rule $9, 10$ and Definition 5.9 we have: $Q(x) \Rightarrow \bigvee_{i=1}^{n} (\exists\, i.G(ev_i) \wedge \neg Q(x))$. This predicate is the form of DLF proof obligation generated from Event-B machine to prove machine FR_M2 is deadlock-free in Q(x)(**).

From (*) and (**), based on the reasoning technique in [71], we have a conclusion: $\{FR\} \vdash \Box\Diamond\neg Q(x)$. ■

5.5 A case study: Container Crane Control

In this section, first we introduce a scenario of Container Crane Control [78], then follow the approach in Section 5.3 to model this system in several refinement steps.

5.5.1 Scenario description

Container cranes are used to load and unload containers on a off ships in most harbors. They pick up single containers with cables that are mounted on the crane head (Figure 5.4).

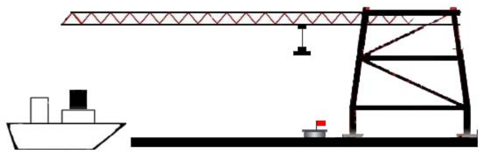


FIGURE 5.4: Container Crane Control system

The crane head moves on a horizontal track from a starting position. The speed of the crane head is controlled by a motor power with a speed level. We start the motor with a fast speed. If the crane head is still far away from the container, we adjust the motor power to a medium speed. If the crane head is in a distance nearer to the target, we reduce the speed to slow. When the container is close to the target position, the speed should be very slow. When the container is above the container, we stop the motor. The crane head loads containers and goes back to the start position.

From this description of the system, a collection of fuzzy requirements FR is extracted as follows:

FR_1. if the crane is at starting position, then power is fast level

FR_2. if the distance to the container is far, then power is medium level

FR_3. if the distance to container is medium, then power is adjusted to slow level

FR_4. if the distance is close, then power is very slow level

FR_5. if the crane is above the container, then power is stopped.

The system has a safety property such that the speed of motor can not be high if the target is not far (property $\mathcal{I}$). The system needs to satisfy that the crane head eventually is above the container from the start position (property $\mathcal{Q}$). Then we have to check if $\{FR\} \vdash \mathcal{I}$ and $\{FR\} \vdash \Box\Diamond\mathcal{Q}$.

5.5.2 Modeling the Crane Container Control system

In this Section, we will use Event-B to model the Container Crane Control system as the method proposed in Section 5.3. First, we model the discrete behavior and the continuous one. Then the eventuality properties will be formalized.

5.5.2.1 Modeling discrete behavior

Applying the translation rules presented in Subsection 5.3.2, we first translate the set of requirements to the definition of Event-B context as follows:

- Apply *Rule 1*: Fuzzy hedges, generators and values in the collection of requirements are translated into the sets HEDGES, DISTANCE and POWER in an Event-B context $Crane\_C0$.
- Apply *Rule 2*: The degree membership functions of hedges and fuzzy values are presented as natural number-valued functions. For example: $h\_deg : HEDGES \rightarrow \mathbb{N}$ states one of hedges. We have another axiom for this function such as $h\_deg(very) = 3 \wedge h\_deg(quite) = 2 \wedge h\_deg(precise) = 1$.

Context $Crane\_C0$ is presented partially as follows

CONTEXT Crane_C0

SETS

HEDGES, DISTANCE, POWER

CONSTANTS

fast, slow, zero, very, quite, precise, start, far, medium, close, above

AXIOMS

axm1 : partition(HEDGES, {very}, {quite}, {precise})

axm2 : partition(DISTANCE, {start}, {far}, {medium}, {close}, {above})

axm3 : partition(POWER, {fast}, {slow}, {zero})

END

We continue to formalize the dynamic part of the model with the following translations.

- Apply *Rule 3*: Linguistic variables in the requirements are translated into Event-B constructs such as *distance* and *power*. Types of these two variables are represented by invariants $inv1$ and $inv2$.
- Apply *Rule 4*: Each imprecise requirement FR_i of the system is translated to an EVENT evt_i, $i = \overline{1,5}$. More specifically, the imprecise requirement $r4$ is translated to evt_4 illustrated in the machine $Crane\_M0$. The other requirements are translated similarly. Moreover, in the initial states, *distance* is equal to *start* and *power* is stopped (modeled in Initialisation event).

The machine $Crane\_M0$ is described partially as follows:

MACHINE Crane_M0

SEES Crane_C0

VARIABLES

power

distance

INVARIANTS

inv1 : power $\in \mathcal{P}$ (HEDGES $\times$ POWER)

inv2 : distance $\in \mathcal{P}$ (HEDGES $\times$ DISTANCE)

```
    inv3 : ran(distance) = close ⇒ ¬ran(power) = {fast}
EVENTS
Initialisation
    begin
        act1 : distance := {precise ↦ start}
        act2 : power := {precise ↦ zero}
    end
Event  evt4 ≙
Status anticipated
    when
        grd1 : distance = {precise ↦ close}
    then
        act1 : power := {very ↦ slow}
        act2 : distance := {precise ↦ above}
    end
END
```

5.5.2.2 First Refinement: Modeling continuous behavior

We refine the translated model in the first part of this section by having a closer view of the system. In fact, each movement of the Crane head is attaching to time axis as it is moving continuously while the power is adjusted discontinuously. We apply rules presented in Section 5.3.3 as follows:

- Apply *Rule 6*: Five events are refined in the refined machine *Crane_M*1, variable t (time counter) is added.

- Apply *Rule 7*: Refines *dis* by dis_c (the distance which is time-dependent). The new variable of refined machine dis_c and one of abstract machine *dis* have a gluing variant ($inv3$).

Refined machine *Crane_M1* for continuous behaviour is partly described as follows:

MACHINE Crane_M1
REFINES Crane_M0
VARIABLES

    t
    dist_c

INVARIANTS

    inv1 : $\mathtt{dist_c} \in \mathbb{R}^+ \rightarrow \mathtt{DISTANCE}$
    inv2 : $\mathtt{t} \in \mathtt{dom(dist\_c)}$
    inv3 : $\mathtt{dist\_c(t)} = \mathtt{distance}$

EVENTS
Initialisation

    begin
        act1 : $\mathtt{t} := 0$
        act2 : $\mathtt{dist\_c} := \{0 \mapsto (\mathtt{precise} \mapsto \mathtt{start})\}$
    end

Event *evt4* $\widehat{=}$
Status anticipated

    when
        grd1 : $\mathtt{dist\_c(t)} = \mathtt{precise} \mapsto \mathtt{medium}$
    then
        act3 : $\mathtt{t} := \mathtt{t} + 1$

act4 : $\mathtt{dist\_c} := \mathtt{dist\_c} \cup \{\mathtt{t} + 1 \mapsto (\mathtt{precise} \mapsto \mathtt{close})\}$

end

END

5.5.2.3 Second Refinement: Modeling eventuality property

We perform the refinement strategy by following the method described in Section 5.4.4 to model the desired eventuality property. First, we apply Rule 6 to extend the abstract context *CraneCtrl_C*0 to define CraneCtrl_C1 by introducing three total functions for numerical values of fuzzy sets. The specification of this context is partially described as follows:

CONTEXT Crane_C1

EXTENDS Crane_C0

CONSTANTS

deg_HED, deg_POWER, d_DIS

AXIOMS

axm4 : $\mathtt{deg\_HED} : \mathtt{HEDGES} \rightarrow \mathbb{N}$

axm5 : $\mathtt{deg\_HED}(\mathtt{very}) = 3 \land \mathtt{deg\_HED}(\mathtt{quite}) = 2$
$\land\ \mathtt{deg\_HED}(\mathtt{precise}) = 1$

END

We refine the abstract machine Crane_M0 to have Crane_M1 with five convergent events (following Rule 7). The snippets below show *evt*4 only.

MACHINE Crane_M1

REFINES Crane_M0

SEES Crane_C1

VARIABLES

d

VARIANT

d

INVARIANTS

inv1 : $d \in \mathbb{N}$

DELF : $d = 0 \Rightarrow ran(distance) = \{start\} \lor ran(distance) = \{far\} \lor ran(distance) = \{medium\} \lor ran(distance) = \{close\} \lor ran(distance) = \{above\}$

EVENTS

Event *evt4_CE* $\widehat{=}$

Status convergent

extends evt_4

when

grd1 : $distance = \{precise \mapsto close\}$

grd2 : $d = deg\_DIS(close)$

grd3 : $\neg d = deg\_DIS(above)$

then

act1 : $power := \{very \mapsto slow\}$

act2 : $distance := \{precise \mapsto above\}$

act2 : $d := deg\_DIS(above)$

end

Event *evt4_OE* $\widehat{=}$

Status ordinary

extends evt_4

 when

  grd1 : $\mathtt{distance} =$

   $\{\mathtt{precise} \mapsto \mathtt{close}\}$

  grd2 : $\mathtt{d} = \mathtt{deg\_DIS}(\mathtt{close})$

  grd3 : $\mathtt{d} = \mathtt{deg\_DIS}(\mathtt{above})$

 then

  act1 : $\mathtt{power} := \{\mathtt{very} \mapsto \mathtt{slow}\}$

  act2 : $\mathtt{distance} :=$

   $\{\mathtt{precise} \mapsto \mathtt{above}\}$

  act2 : $\mathtt{d} := \mathtt{deg\_DIS}(\mathtt{above})$

 end

END

5.5.3 Checking properties

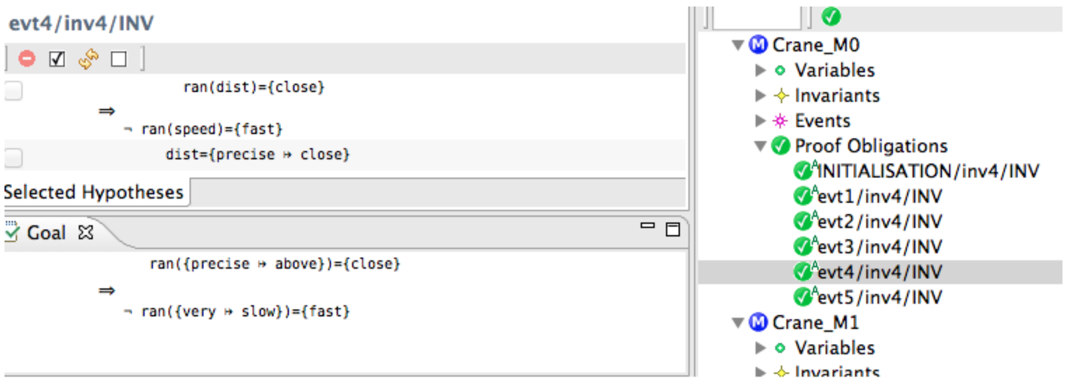
The system has a safety property which is formalized as an invariant clause
$inv4 : ran(distance) = \{close\} \Rightarrow \neg ran(power) = \{fast\}$. Invariant preservation PO is generated for each event of the machine $Crane\_M0$. Table 5.1 shows the invariant preservation PO for invariant $inv4$ of event $evt4$

TABLE 5.1: INV PO of event $evt4$

$ran(distance) = \{close\} \Rightarrow \neg ran(speed) = \{fast\}$ $dis = \{precise \mapsto close\}$ $\vdash$ $ran\,(\{precise \mapsto above\}) = \{close\} \Rightarrow \neg ran\,(\{very \mapsto slow\}) = \{fast\}$	$evt4/inv4/INV$

All such proof obligations are generated and discharged automatically for machine $Crane\_M0$ using the Rodin tool under the label $evt_i/inv4/INV$, $i =$

$\overline{1..5}$ (Figure 5.5). It ensures that invariant is preserved through events, i.e. the collection requirements of this system conform to the safety property.


evt4/inv4/INV

```
ran(dist)={close}
⇒
¬ ran(speed)={fast}
dist={precise ⤀ close}
```

Selected Hypotheses

Goal

```
ran({precise ⤀ above})={close}
⇒
¬ ran({very ⤀ slow})={fast}
```

Crane_M0
Variables
Invariants
Events
Proof Obligations
INITIALISATION/inv4/INV
evt1/inv4/INV
evt2/inv4/INV
evt3/inv4/INV
evt4/inv4/INV
evt5/inv4/INV
Crane_M1
Variables
Invariants

FIGURE 5.5: Safety properties are ensured in the Rodin tool automatically

While safety property is maintained in every refinement, eventuality can only be verified in the machine $Crane\_M1$. Hence, we have to prove that eventually the crane loader will reach above position of container, i.e. $Crame\_M1 \vdash \Box\Diamond(d = deg\_DIS(above))$. The deadlock-free property of this machine is encoded as the theorem $DELF$ in $Crane\_M1$. Its proof obligation is generated as $DELF/THM$.

TABLE 5.2: Deadlock free PO of machine $Crane\_M1$

$d = deg\_DIS(above)$ $\Rightarrow$ $d = deg\_DIS(start) \lor d = deg\_DIS(far)$ $d = deg\_DIS(medium) \lor d = deg\_DIS(close)$ $d = deg\_DIS(above)$	DELF/THM

In order to check the convergent property, proof obligations are generated for each convergent events of machine $Crane\_M1$ (evt_i/NAT and evt_i/VAR). Table 5.3 is the proof obligation that shows event $evt4$ of machine $Crame\_M1$ decrease variant d.

All proof obligations of the deadlock-freeness and convergence are discharged automatically in the Rodin tool.

Table 5.3: VAR PO of event *evt4*

$dis = \{precise \mapsto close\}$ $\neg d = deg\_DIS(close)$ $d = deg\_DIS(close)$ $\vdash$ $d - (deg\_DIS(close) - deg\_DIS(above)) < d$	evt4_CE/VAR

5.6 Chapter conclusions

There are few work up to date that have addressed the problem of modeling and verifying systems described by imprecise requirements. Most of research results have focused on modeling and representing fuzzy terms. In this chapter, we use classical set theory to represent fuzzy terms, after that formalize and verify it by an Event-B model. Since formal methods has been researched so far to dedicatedly formalize precise requirements, our work provides a novel approach to model imprecise requirements. We extend Fuzzy If-Then rules by timed Fuzzy If-Then rules which can be used to describe the continuous behavior of the system. This chapter also presents a new refinement-based method for modeling and verifying safety and eventuality properties of such systems. As far as we know, the proposed methods provide the first concrete results of formal checking such properties of imprecise requirements. The research result of checking safety property was published in [79] and the one of checking eventuality properties are accepted to appear in *Proceedings of The 30th ACM/SIGAPP Symposium On Applied Computing - SE Track* [80].

However, due to some limitation of the RODIN, we had to introduce a kind of approximation to use $\mathbb{N}$ instead of $\mathbb{R}$. This limitation will be overcome by incorporating new plugins [81]. One drawback of our method is that the eventualities can only be checked at the last refinement. To overcome limitation, we need to improve the reasoning methods.

Chapter 6

Conclusions

6.1 Achievements

Constructing reliable software systems is one of the most important objectives of software engineering. Using formal methods such as formal modeling and verification is one of the most recommended ways. It allows to have not only a better understand of the system but also can perform formal reasoning about the correctness of significant properties. Formal modeling and verification techniques ensure the absent of errors mathematically. For each kind of software architectures and styles, ones propose suitable methods for modeling and verifying.

The event-driven architecture is applied widely for many software systems in various domains. The event-driven systems are able to send and react to events to outside or to different inside components of the systems. Hence, it looses the coupling between the system's components. One important benefit of this architectural style is that it provides strong support for reuse. Any component can be introduced into a system simply by registering it for the events of that system. Another advantage is that its components may be replaced easily by other components without affecting the interfaces of other components in the system. Besides these advantages, the event-driven systems may face many problems. For example,when a component announce an event, the system does have any registered component to handle that or can not know if it is finished. Reasoning about correctness of such systems also can be problematic, since the meaning of a procedure that announces events will depend on the context of bindings in which it is invoked. As a consequence, formal modeling and verification of such systems are an emerging topic that inspires many research group in the world. It is valuable especially if we can verify the system at early design time because it reduces the cost of development. Many research work have been attempted for this topic but it is still insufficient.

In comparison to the previous work, the thesis provides a different approach in modeling and verifying even-driven systems. The thesis covers not only the systems described by precise requirements but also the imprecise ones. The thesis proposes new methods that use Event-B formal method for analyzing such systems. Event-B is mainly based on first order logic, set theory and is suitable for complex, large reactive systems. Event-B communities also provide rich of supporting tools to model and verify the software system.

In the first part of the thesis, instead of working on a reference model of event-driven architecture which are more abstract and describe a larger class of systems, we focus on applications of two types of even-driven systems database systems including triggers and context-ware systems. Two applications have particular properties and provided functionalities. Though, in these systems, triggers and production rules have the same structure which is in the form of ECA format. Our proposed methods are based on the similar working mechanism of an ECA rule and an Event-B events. For this reason, the modeling process is natural and easy. Furthermore, since we directly use Event-B to formalize the systems, we do not need any more intermediate step to check the system correctness. We summarize these methods as follows

- To analyze modern database systems including active databases, we propose a set of translation rules to translate all objects of database systems to Event-B constructs. Formalization of database triggers is deeply investigated. The significant properties such as business constraints, and looping are can be expressed and checked by invariants clauses. The supporting tool Trigger2B is also developed to assist the automatic translation of database systems to Event-B models. This result may overcome the difficulty that makes formal methods absent in real database applications development. The practical tool is supposed to automatically translate and export the result to file formats that is understandable by several provers such as the Rodin. Then database developers can easily check the correctness of the system that they are designing.

- To analyze context-aware systems, we map the concept of context data to Event-B context. The relations of context entities hence can be expressed by sets, axioms and Event-B context extension mechanism. The other components of a context-aware system is also mapped to Event-B concepts. The proposed method makes use of Event-B refinement to model the system incrementally. The important property such as the context constraint can be verified in every refinement stage.

In the second part, the thesis also makes significant contributions on analyzing event-driven systems specified by imprecise requirements. Although imprecise requirements are often found in software development processes, few work have been addressed the problem of modeling and verifying such descriptions so far. This part presents a new specification and verification framework, in which the requirements were modeled in the Fuzzy If-Then rules. The rules were translated into a set of Event-B descriptions so that the refinement-based modeling method could be applied for the verification. We summarize this part as follows

- Modeling: We first propose to use classical-set to represent fuzzy sets, and prove that a set of Fuzzy If-Then rules can be represented by classical set. We also work on modeling of timed fuzzy event-driven systems by introducing a new form of timed Fuzzy If-Then rules. This modeling method allows to formalize both discrete and timed systems which are described by imprecise requirements that is often being used in fuzzy control systems.
- Verification: The proposed methods make use of Event-B refinement and use existing reasoning methods to formally verify both safety and eventuality properties. These are the first concrete results of formal checking of such properties for imprecise system requirements.

The illustrative examples of the thesis are modeled by the Rodin tool which allows to automatically generate necessary proof obligations for verification. Almost generated POs are also discharged automatically in the Rodin tool. This helps to reduce the complexity and effort of manual proving for proof obligations.

6.2 Limitations

Besides the achievement, the thesis still has remaining issues need to be discussed. There are many types of event-driven systems including graphic user interfaces of programs where user interface events signify program commands, rule-based production systems where a condition becoming true causes an action to be triggered, and active objects where changing a value of an object's attribute triggers some actions. Scope of the thesis is handling with two applications of such as database systems including triggers and context-aware systems. Furthermore, the event control mechanism also has various kinds but these systems only use ECA mechanism to describe responsive actions to raised events. Using If-Then rules and Fuzzy If-Then rules to express the behavior of the system seem not enough in some cases. We discuss relating issues for each proposed method as follows

- The proposed method for modeling and verifying database systems does not support to reason directly about termination property, while it is one of desired properties that developers want to check. It also just handle is simple case that contains only a sequence of DML statements that does not contain nested statements and full trigger syntax such as for/loop statements. In case that we want to formalize any kind of triggers, we need to propose more efficient algorithms to parse and translate their content. Moreover, this thesis also just handle with DML triggers but do not consider other types of triggers.
- The proposed method for modeling context-aware systems already reuse Event-B concept to represent context data. Due to lacking of primitive data type support in Event-B, we can only enrich context

data modeling by incorporating new plugins. Context data is often complex and contains many types of data. Furthermore, a real context-ware application often contains time related data. However, Event-B does not support temporal logic, hence modeling and verifying such applications will face several problems. The proposed method needs to be extended to model time dependent variables.

- The method for modeling and verifying imprecise systems requirements handles both cases of discrete and continuous behavior of the systems. It analyse both safety and eventuality properties of the systems. We showed that the verification was mostly conducted automatically using the current RODIN tool. However, due to some limitation of the RODIN, we had to introduce a kind of approximation to use $\mathbb{N}$ instead of $\mathbb{R}$. Moreover, time related properties are not discussed yet. Describing the behavior of the system by Fuzzy If-Then rules is also not general enough. Besides eventuality properties, there are several liveness properties are necessary to be verified to warranty the system correctness such as progress, persistence. These kinds of properties are not mentioned yet.

6.3 Future work

In the future, we continue to research further on the topics which have been presented in the thesis and have been achieved with the beginning results. More specifically, the future research directions are as follows

- One of the thesis research direction is developing a Rodin plugin tool for database trigger systems modeling. We also will handle more complex triggers with nested DML statements combining with loop and condition statements. In case of complex nested statements, we may need to apply composition techniques to model that kind of triggers by composited events. Reasoning about termination property of triggers is going to investigated along with considering more types of triggers is one of our future work.

- We will extend the method for modeling method context-aware systems by using the Theory plugin which allows to create and define semantics for various kind of context data which are frequently used such as: time, location. The proposed method will be extended to modeling more complex relationship between contexts. Currently, there are several framework for describe context-aware. We intend to directly map context specification language to Event-B.

- With proposed method, a collection of imprecise requirements which are described by Fuzzy If-Then rules can be specified by Event-B. It introduced a concept of timed Fuzzy If-Then rules to model timed systems but it is not investigated deeply yet. For example, the verification of the interesting properties which are time-dependent is not discussed yet. Our future work in this direction will focus on analyzing such properties.

- The current method for proving liveness properties is implemented at the last refinement. The work for other interesting liveness properties. Therefore, proving liveness properties at every refinement stage is also an objective. Furthermore, the theoretical background for liveness reasoning in Event-B also need to be extended for general cases including fairness assumption. That also makes it possible to verify the other important liveness properties such as persistence, progress.

LIST OF PUBLICATIONS

1. Hong Anh Le and Ninh Thuan Truong. Modeling and Verifying WS-CDL Using Event-B. In Proc. ICCASA 2012. LNICST Vol 109, pp. 290-299, Springer, 2013.

2. Hong Anh Le and Ninh Thuan Truong: Modeling and Verifying DML Triggers Using Event-B, In Proc. ACIIDS 2013. LNCS Vol 7083, Vol 2, pp. 539-548, Spinger, 2013.

3. Hong Anh Le, Loan Dinh Thi and Ninh Thuan Truong: Modeling and Verifying Imprecise Requirements of Systems Using Event-B. In Proc. KSE 2013. AISC Vol 244, pp. 313-325, Springer, 2013.

4. Hong Anh Le and Ninh Thuan Truong: Formal Modeling and Verification of Context-Aware Systems Using Event-B. In Proc. ICCASA 2013. LNICST Vol 128, pp. 250-259, Springer 2014 (**The best paper award**).

5. Hong Anh Le and Ninh Thuan Truong: Formal Modeling and Verification of Context-Aware Systems Using Event-B. In EAI Endorsed Transactions on Context-Aware Systems and Applications. ISSN 2409-0026. (accepted)

6. Hong Anh Le, Ninh Thuan Truong and Shin Nakajima: Verifying Eventuality Properties of Imprecise System Requirements. The 30th ACM/SIGAPP Symposium On Applied Computing - Software Engineering Track, April 13–17, 2015. Salamanca, Spain. (accepted)

Bibliography

[1] Kevin Lano. *The B Language and Method: A Guide to Practical Formal Development*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1996. ISBN 3540760334.

[2] Harry Chen. *An Intelligent Broker for Context-Aware Systems*. PhD thesis, University of Maryland, 2004.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008. ISBN 026202649X, 9780262026499.

[4] Melvin Fitting. *First-order Logic and Automated Theorem Proving (2Nd Ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996. ISBN 0-387-94593-8.

[5] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999. ISBN 3540654100.

[6] Ian Sommerville. *Software Engineering (8th Ed.)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2007. ISBN 978-0-321-31379-9.

[7] Gabor Madl. *Model-based Analysis of Event-driven Distributed Real-time Embedded Systems*. PhD thesis, Long Beach, CA, USA, 2009. AAI3364975.

[8] J.M. Atlee and J. Gannon. State-based model checking of event-driven system requirements. *Software Engineering, IEEE Transactions on*, 19(1):24–40, Jan 1993. ISSN 0098-5589. doi: 10.1109/32.210305.

[9] I. Ray and P. Ammann. Using the b-toolkit to ensure safety in scr specifications. In *Computer Assurance, 1997. COMPASS '97. Are We Making Progress Towards*

Computer Assurance? Proceedings of the 12th Annual Conference on, pages 1–12, Jun 1997.

[10] Ludger Fiege, Gero Mühl, and Felix C. Gärtner. Modular event-based systems. *Knowl. Eng. Rev.*, 17(4):359–388, December 2002. ISSN 0269-8889. doi: 10.1017/S0269888903000559. URL http://dx.doi.org/10.1017/S0269888903000559.

[11] Huy Tran and Uwe Zdun. Event actors based approach for supporting analysis and verification of event-driven architectures. In *Proceedings of the 2013 17th IEEE International Enterprise Distributed Object Computing Conference*, EDOC '13, pages 217–226, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-5081-7. doi: 10.1109/EDOC.2013.32. URL http://dx.doi.org/10.1109/EDOC.2013.32.

[12] Muffy Calder and Michele Sevegnani. Process algebra for event-driven runtime verification: A case study of wireless network management. In John Derrick, Stefania Gnesi, Diego Latella, and Helen Treharne, editors, *Integrated Formal Methods*, volume 7321 of *Lecture Notes in Computer Science*, pages 21–23. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-30728-7. doi: 10.1007/978-3-642-30729-4_2. URL http://dx.doi.org/10.1007/978-3-642-30729-4_2.

[13] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010. ISBN 0521895561, 9780521895569.

[14] J.-R. Abrial. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996. ISBN 0-521-49619-5.

[15] Dblp. http://dblp.org, 2014. URL http://dblp.org.

[16] Event-b and the rodin platform. http://www.event-b.org, 2012.

[17] Ken Robinson. The b method and the b toolkit. In Michael Johnson, editor, *Algebraic Methodology and Software Technology*, volume 1349 of *Lecture Notes in Computer Science*, pages 576–580. Springer Berlin Heidelberg, 1997. ISBN 978-3-540-63888-9. doi: 10.1007/BFb0000503. URL http://dx.doi.org/10.1007/BFb0000503.

[18] C. M. Prashanth, K.C. Shet, and J. Elamkulam. An efficient event based approach for verification of uml statechart model for reactive systems. In *Advanced Computing and Communications, 2008. ADCOM 2008. 16th International Conference on*, pages 357–362, Dec 2008.

[19] S. Jalili and M. Mirzaaghaei. Rverl: Run-time verification of real-time and reactive programs using event-based real-time logic approach. In *Software Engineering Research, Management Applications, 2007. SERA 2007. 5th ACIS International Conference on*, pages 550–557, Aug 2007. doi: 10.1109/SERA.2007.116.

[20] Marcelo d'Amorim and Klaus Havelund. Event-based runtime verification of java programs. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, May 2005. ISSN 0163-5948. doi: 10.1145/1082983.1083249. URL http://doi.acm.org/10.1145/1082983.1083249.

[21] José Luiz Fiadeiro and Antónia Lopes. An algebraic semantics of event-based architectures. *Mathematical. Structures in Comp. Sci.*, 17(5):1029–1073, October 2007. ISSN 0960-1295. doi: 10.1017/S0960129507006299.

[22] E. Posse and J. Dingel. Kiltera: A language for timed, event-driven, mobile and distributed simulation. In *Distributed Simulation and Real Time Applications (DS-RT), 2010 IEEE/ACM 14th International Symposium on*, pages 87–96, Oct 2010. doi: 10.1109/DS-RT.2010.19.

[23] E. Posse and H. Vangheluwe. Kiltera: A simulation language for timed, dynamic structure systems. In *Simulation Symposium, 2007. ANSS '07. 40th Annual*, pages 293–300, March 2007. doi: 10.1109/ANSS.2007.25.

[24] Aymen Baouab, Olivier Perrin, and Claude Godart. An event-driven approach for runtime verification of inter-organizational choreographies. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 640–647, July 2011.

[25] G.L.J.M. Janssen. Hardware verification using temporal logic: A practical view. In L. Claesen, editor, *Formal VLSI Correctness Verification*, pages 159–168. Elsevier Science Publisher (North-Holland), 1990.

[26] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer, 2012.

[27] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[28] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[29] L.A. Zadeh. Toward a theory of fuzzy systems. Technical Report NASA CR-1432, NATIONAL AERONAUTICS AND SPACE ADMINISTRATION, 1969.

[30] Jeannette M. Wing. A specifier's introduction to formal methods. *Computer*, 23 (9):8–23, September 1990. ISSN 0018-9162.

[31] John V. Guttag and James J. Horning. *Larch: Languages and Tools for Formal Specification*. Springer-Verlag New York, Inc., New York, NY, USA, 1993. ISBN 0-387-94006-5.

[32] Kokichi Futatsugi, Joseph A. Goguen, Jean-Pierre Jouannaud, and José Meseguer. Principles of obj2. In *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '85, pages 52–66, New York, NY, USA, 1985. ACM. ISBN 0-89791-147-4. doi: 10.1145/318593.318610. URL http://doi.acm.org/10.1145/318593.318610.

[33] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. ISBN 0-13-983768-X.

[34] Cliff B. Jones. *Systematic Software Development Using VDM (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990. ISBN 0-13-880733-7.

[35] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8): 666–677, August 1978. ISSN 0001-0782. doi: 10.1145/359576.359585. URL http://doi.acm.org/10.1145/359576.359585.

[36] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981. ISBN 0136619835.

[37] Jan Storbank Pedersen Mark H. Klein. Using the vienna development method (vdm) to formalize a communication protocol. Technical report, Software Engineering Institute Carnegie Mellon University, 1998.

[38] Atelier b. http://www.atelierb.eu/en/, 2013. URL http://www.atelierb.eu/en/.

[39] Andrew Eisenberg and Jim Melton. Sql: 1999, formerly known as sql3. *SIGMOD Rec.*, 28(1):131–138, March 1999. ISSN 0163-5808. doi: 10.1145/309844.310075. URL http://doi.acm.org/10.1145/309844.310075.

[40] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

[41] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66550-1. URL http://dl.acm.org/citation.cfm?id=647985.743843.

[42] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, jun 2007. ISSN 1743-8225.

[43] S.-Y. Lee and T.-W. Ling. Are your trigger rules correct? In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, DEXA '98, pages 837–, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8353-8.

[44] Sin Yeung Lee and Tok Wang Ling. Verify updating trigger correctness. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, DEXA '99, pages 382–391, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-66448-3.

[45] Eun-Hye Choi, Tatsuhiro Tsuchiya, and Tohru Kikuno. Model checking active database rules under various rule processing strategies. *IPSJ Digital Courier*, 2: 826–839, 2006.

[46] Lorena Chavarría-Báez and Xiaoou Li. Verification of active rule base via conditional colored petri nets. In *SMC*, pages 343–348, 2007.

[47] Indrakshi Ray and Indrajit Ray. Detecting termination of active database rules using symbolic model checking. In *Proceedings of the 5th East European Conference on Advances in Databases and Information Systems*, ADBIS '01, pages 266–279, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42555-1.

[48] Elena Baralis. Rule analysis. In *Active Rules in Database Systems*, pages 51–67. Springer, New York, 1999.

[49] Tarek Ghazi and Michael Huth. An Abstraction-Based Analysis of Rule Systems for Active Database Management Systems. Technical report, Kansas State University, April 1998. Technical Report KSU-CIS-98-6, pp15.

[50] Eun-Hye Choi, Tatsuhiro Tsuchiya, and Tohru Kikuno. Model checking active database rules. Technical report, AIST CVS, Osaka University, Japan, 2006.

[51] R.Manicka Chezian and T.Devi. A new algorithm to detect the non-termination of triggers in active databases. *International Journal of Advanced Networking and Applications*, 3(2):1098–1104, 2011.

[52] Antlr v3. http://www.antlr3.org, 2012.

[53] HongAnh Le and NinhThuan Truong. Modeling and verifying dml triggers using event-b. In Ali Selamat, editor, *Intelligent Information and Database Systems*, volume 7803 of *Lecture Notes in Computer Science*, pages 539–548. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36542-3.

[54] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[55] Claudia Linnhoff-Popien Michael Samulowitz, Florian Michahelles. Capeus: An architecture for context-aware selection and execution of services. In Krzysztof Zieliński, Kurt Geihs, and Aleksander Laurentowski, editors, *New Developments*

in Distributed Applications and Interoperable Systems, volume 70 of *IFIP International Federation for Information Processing*, pages 23–39. Springer US, 2002. ISBN 978-0-7923-7481-7. URL http://dx.doi.org/10.1007/0-306-47005-5_3.

[56] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Proceedings of the First International Conference on Pervasive Computing*, Pervasive '02, pages 167–180, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44060-7.

[57] Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, and Karen Henricksen. Experiences in using cc/pp in context-aware systems. In *Proceedings of the 4th International Conference on Mobile Data Management*, MDM '03, pages 247–261, London, UK, UK, 2003. Springer-Verlag. ISBN 3-540-00393-2. URL http://dl.acm.org/citation.cfm?id=648060.747270.

[58] S.K. Mostefaoui. A context model based on uml and xml schema representations. In *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on*, pages 810–814, 2008.

[59] MohamedSalah Benselim and Hassina Seridi-Bouchelaghem. Extended uml for the development of context-aware applications. In Rachid Benlamri, editor, *Networked Digital Technologies*, volume 293 of *Communications in Computer and Information Science*, pages 33–43. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-30506-1. URL http://dx.doi.org/10.1007/978-3-642-30507-8_4.

[60] Anjum Shehzad, Hung Q. Ngo, Kim Anh Pham, and S. Y. Lee. Formal modeling in context aware systems. In *In Proceedings of The 1 st International Workshop on Modeling and Retrieval of Context (MRC'2004)*, 2004.

[61] D. Ejigu, M. Scuturici, and L. Brunie. An ontology-based approach to context modeling and reasoning in pervasive computing. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on*, pages 14–19, 2007.

[62] Mikkel B. Kjaergaard and Jonathan Bunde-Pedersen. Towards a formal model of context awareness. In *First International Workshop on Combining Theory and Systems Building in Pervasive Computing 2006 (CTSB 2006)*, 2006. URL http://www.daimi.au.dk/~{}jbp/pmwiki.uploads//conawa.baun.bunde-pedersen.pdf.

[63] Minh H. Tran, Alan Colman, Jun Han, and Hongyu Zhang. Modeling and verification of context-aware systems. In *Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference - Volume 01*, APSEC '12, pages 79–84, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4922-4.

[64] Alan W. Colman. *Role oriented adaptive design*. PhD thesis, Swinburne University of Technology, 2006.

[65] HongAnh Le and NinhThuan Truong. Formal modeling and verification of context-aware systems using event-b. In Phan CV, editor, *Context-Aware Systems and Applications*, volume 128 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 250–259. Springer International Publishing, 2014. ISBN 978-3-319-05938-9. doi: 10.1007/978-3-319-05939-6_25.

[66] HongAnh Le and NinhThuan Truong. Formal modeling and verification of context-aware systems using event-b. *EAI Endorsed Transactions on Context-aware Systems and Applications*. ISSN 2409-0026. (accepted).

[67] Benedetto Intrigila, Daniele Magazzeni, Igor Melatti, and Enrico Tronci. A model checking technique for the verification of fuzzy control systems. In *Proc. CIMCA-IAWTIC'06 - Volume 01*, CIMCA '05, pages 536–542, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2504-0-01.

[68] Stephen J. H. Yang, Jeffrey J. P. Tsai, and Chyun-Chyi Chen. Fuzzy rule base systems verification using high-level petri nets. *IEEE Trans. Knowl. Data Eng.*, 15(2):457–473, 2003.

[69] Chris Matthews and Paul A. Swatman. Fuzzy concepts and formal methods: A fuzzy logic toolkit for z. In *Proceedings of the First International Conference of B*

and Z Users on Formal Specification and Development in Z and B, ZB '00, pages 491–510, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67944-8.

[70] C. Matthews and P. A. Swatman. Fuzzy concepts and formal methods: some illustrative examples. In *Proc. of APSEC 2000*, APSEC '00, pages 230–238, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0915-0.

[71] ThaiSon Hoang and Jean-Raymond Abrial. Reasoning about liveness properties in event-b. In *Formal Methods and Software Engineering*, volume 6991 of *LNCS*, pages 456–471. 2011. ISBN 978-3-642-24558-9.

[72] Viktor Pavliska. Petri nets as fuzzy modeling tool. Technical report, University of Ostrava - Institute for Research and Applications of Fuzzy Modeling, 2006.

[73] Jaroslav Knybel and Viktor Pavliska. Representation of fuzzy if-then rules by petri nets. In *ASIS 2005*, pages 121–125, Prerov. Ostrava, Sept 2005.

[74] Jonathan Lee, Nien-Lin Xue, Kuo-Hsun Hsu, and Stephen J. Yang. Modeling imprecise requirements with fuzzy objects. *Inf. Sci.*, 118(1-4):101–119, September 1999. ISSN 0020-0255.

[75] J. Lee, Yong-Yi FanJiang, Jong-Yin Kuo, and Ying-Yan Lin. Modeling imprecise requirements with xml. In *Proc. of FUZZ-IEEE'02*, volume 2, pages 861–866, 2002.

[76] Marlene Goncalves, Rosseline Rodríguez, and Leonid Tineo. Formal method to implement fuzzy requirements. *RASI*, 9(1):15–24, 2012.

[77] Jean-Raymond Abrial, Wen Su, and Huibiao Zhu. Formalizing hybrid systems with event-b. In *Proc. ABZ 2012*, volume 7316 of *LNCS*, pages 178–193. 2012. ISBN 978-3-642-30884-0. URL http://dx.doi.org/10.1007/978-3-642-30885-7_13.

[78] Fuzzytech home page, 2012. http://www.fuzzytech.com.

[79] HongAnh Le, LoanDinh Thi, and NinhThuan Truong. Modeling and verifying imprecise requirements of systems using event-b. In Huynh VN, editor, *Knowledge and Systems Engineering*, volume 244 of *Advances in Intelligent Systems and Computing*, pages 313–325. Springer International Publishing, 2014.

[80] Ninh Thuan Truong Hong Anh Le and Shin Nakajima. Verifying eventuality properties of imprecise system requirements using event-b. In *The 30th ACM/SIGAPP Symposium On Applied Computing - Software Engineering Track*, April 2015. (accepted).

[81] Michael Butler and Issam Maamria. Practical theory extension in event-b. In *Theories of Programming and Formal Methods*, volume 8051, pages 67–81. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39697-7. doi: 10.1007/978-3-642-39698-4_5. URL http://dx.doi.org/10.1007/978-3-642-39698-4_5.

[82] Idir Ait-Sadoune and Yamine Ait-Ameur. From bpel to event-b. In *IM FMT'09 Conference*, Düsseldorf Germany, Fevruary 2009.

[83] Parnichkun M Aziz M H, Bohez E L and Saha C. Classification of fuzzy petri nets, and their applications. *Engineering and Technology, World Academy of Science*, 72:394–407, 2011.

[84] Elena Baralis and Jennifer Widom. An algebraic approach to static analysis of active database rules. *ACM Trans. Database Syst.*, 25(3):269–332, September 2000. ISSN 0362-5915.

[85] Lorena Chavarría-Báez and Xiaoou Li. A petri net-based metric for active rule validation. In *ICTAI*, pages 922–923, 2011.

[86] Lorena Chavarría-Báez and Xiaoou Li. Ecapnver: A software tool to verify active rule bases. In *ICTAI (2)*, pages 138–141, 2010.

[87] Earl Cox. *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems*. Academic Press Professional, Inc., San Diego, CA, USA, 1994. ISBN 0-12-194270-8.

[88] HongAnh Le and NinhThuan Truong. Modeling and verifying ws-cdl using event-b. In Phan CV, editor, *Context-Aware Systems and Applications*, volume 109 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 290–299. Springer Berlin Heidelberg, 2013.

[89] Xiaoou Li, Joselito Medina Marń, and Sergio Chapa. A structural model of eca rules in active database. In Carlos Coello Coello, Alvaro de Albornoz, Luis Sucar, and Osvaldo Battistutti, editors, *MICAI 2002: Advances in Artificial Intelligence*, volume 2313 of *Lecture Notes in Computer Science*, pages 73–87. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-43475-7.

[90] L.S. Rocha and R.M.C. Andrade. Towards a formal model to reason about context-aware exception handling. In *Exception Handling (WEH), 2012 5th International Workshop on*, pages 27–33, 2012.

[91] Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement, and Proof.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-948472-8.

[92] Yubin Zhong. The design of a controller in fuzzy petri net. *Fuzzy Optimization and Decision Making*, 7:399–408, 2008. ISSN 1568-4539.

[93] Spin. http://spinroot.com/spin, 2013. URL http://spinroot.com/spin.

Appendix A

Event-B specification of Trigger example

This appendix contains full Event-B specification for checking context constraints of Trigger example in Chapter 3.

A.1 Context specification of Trigger example

An Event-B Specification of DB_C
Creation Date: 9Jun2014 @ 04:46:04 PM

CONTEXT DB_C

SETS

```
    TYPES
    TABLE_NAMES
```

CONSTANTS

```
    EMPL
    BONUS
    update
```

insert

delete

TBL_EMPL

TBL_BONUS

AXIOMS

axm1 : partition(TYPES, {update}, {delete}, {insert})

axm2 : partition(TABLE_NAMES, {EMPL}, {BONUS})

axm3 : TBL_EMPL $= \mathbb{N} \times \mathbb{N}$

axm4 : TBL_BONUS $= \mathbb{N} \times \mathbb{N}$

END

A.2 Machine specification of Trigger example

An Event-B Specification of DB_M

Creation Date: 9Jun2014 @ 04:46:04 PM

MACHINE DB_M

SEES DB_C

VARIABLES

pk_empl

pk_bonus

empl_rec

bonus_rec

type

table

INVARIANTS

inv3 : type $\in$ TYPES

inv4 : empl_rec $\in \mathbb{P}($TBL_EMPL$)$

inv5 : bonus_rec $\in \mathbb{P}($TBL_BONUS$)$

inv6 : $\mathtt{table} \in \mathtt{TABLE\_NAMES}$

inv7 : $\forall \mathtt{nid} \cdot \mathtt{nid} \in \mathtt{dom}(\mathtt{empl\_rec}) \land \mathtt{pk\_empl}(\mathtt{nid}) > 5 \Rightarrow \mathtt{pk\_bonus}(\mathtt{nid}) > 5$

inv8 : $\mathtt{pk\_empl} \in \mathtt{dom}(\mathtt{empl\_rec}) \rightarrowtail\!\!\!\!\twoheadrightarrow \mathtt{ran}(\mathtt{empl\_rec})$

inv9 : $\mathtt{pk\_bonus} \in \mathtt{dom}(\mathtt{bonus\_rec}) \rightarrowtail\!\!\!\!\twoheadrightarrow \mathtt{ran}(\mathtt{bonus\_rec})$

inv10 : $\forall \mathtt{nid} \cdot ((\mathtt{nid} \in \mathtt{dom}(\mathtt{empl\_rec}) \land \mathtt{type} = \mathtt{update} \land \mathtt{table} = \mathtt{BONUS} \land \mathtt{pk\_bonus}(\mathtt{nid}) \geq 10) \lor (\mathtt{type} = \mathtt{update} \land \mathtt{table} = \mathtt{EMPL}))$

EVENTS

Initialisation

```
    begin
        act1 : $\mathtt{bonus\_rec} := \{5 \mapsto 10\}$
        act2 : $\mathtt{empl\_rec} := \{5 \mapsto 4\}$
    end
```

Event *trigger1* $\widehat{=}$

```
    any
        eid
    when
        grd1 : $\mathtt{type} = \mathtt{update}$
        grd2 : $\mathtt{table} = \mathtt{EMPL}$
        grd3 : $\mathtt{eid} \in \mathtt{dom}(\mathtt{empl})$
    then
        act1 : $\mathtt{type} := \mathtt{update}$
        act3 : $\mathtt{table} := \mathtt{BONUS}$
        act5 : $\mathtt{bonus} := \{\mathtt{eid} \mapsto (\mathtt{pk\_bonus}(\mathtt{eid}) + 10)\} \oplus \mathtt{bonus}$
        act5 : $\mathtt{pk\_bonus}(\mathtt{eid}) := \mathtt{pk\_bonus}(\mathtt{eid}) + 10$
    end
```

Event *trigger2* $\widehat{=}$

```
    any
        eid
    when
        grd1 : $\mathtt{type} = \mathtt{update}$
        grd2 : $\mathtt{table} = \mathtt{BONUS}$
        grd3 : $\mathtt{pk\_bonus}(\mathtt{eid}) \geq 10$
    then
        act1 : type := update
        act2 : table := EMPL
        act3 : $\mathtt{empl} := \{\mathtt{eid} \mapsto (\mathtt{pk\_empl}(\mathtt{eid}) + 1)\} \oplus \mathtt{empl}$
    end
END
```

Appendix B

Event-B specification of the ACC system

This appendix contains full Event-B specification for checking context constraints of ACC example in Chapter 4.

B.1 Context specification of ACC system

An Event-B Specification of Target
Creation Date: 3Jun2014 @ 10:19:23 AM

CONTEXT Target

CONSTANTS

```
    TARGET_DETECTION
    INIT
    MAX_SPEED
    INC
```

AXIOMS

```
    axm1 : TARGET_DETECTION = BOOL
```

axm2 : $\mathtt{INIT} \in \mathbb{N}$

axm3 : $\mathtt{MAX\_SPEED} \in \mathbb{N}$

axm4 : $\mathtt{INIT} + \mathtt{INC} < \mathtt{MAX\_SPEED}$

axm5 : $\mathtt{INC} \in \mathbb{N}$

END

B.2 Machine specification of ACC system

An Event-B Specification of ACC_M0

Creation Date: 3Jun2014 @ 10:14:52 AM

MACHINE ACC_M0

SEES Target

VARIABLES

speed

target_det

INVARIANTS

inv1 : $\mathtt{speed} \in \mathbb{N}$

inv2 : $\mathtt{target\_det} \in \mathtt{TARGET\_DETECTION}$

inv3 : $\mathtt{speed} \leq \mathtt{MAX\_SPEED}$

EVENTS

Initialisation

begin

act1 : $\mathtt{speed} := \mathtt{MAX\_SPEED}$

end

Event *TargetDetected* $\widehat{=}$

when

grd1 : $\mathtt{target\_det} = \mathtt{TRUE}$

grd2 : $\mathtt{speed} > \mathtt{INC}$

```
    then
        act1 : speed := speed − INC
    end
Event  TargetUndetected ≙
    when
        grd1 : target_det = FALSE
        grd2 : speed < MAX_SPEED − INC
    then
        act1 : speed := speed + INC
    end
END
```

B.3 Extended context

```
CONTEXT  Weather_Road
EXTENDS  Target
CONSTANTS
    RAINING
    SHARP
AXIOMS
    axm1 : RAINING = BOOL
    axm2 : SHARP = BOOL
END
```

B.4 Refined machine

```
MACHINE  ACC_M1
REFINES  ACC_M0
SEES  Weather_Road
```

```
VARIABLES
    isRain
    speed
    target_det
    isSharp
INVARIANTS
    inv1 : $isRain \in RAINING$
    cxt_ct : $isRain = TRUE \lor isSharp = TRUE \Rightarrow speed < MAX\_SPEED$
    inv3 : $isSharp \in SHARP$
EVENTS
Initialisation
    begin
        skip
    end
Event  TargetUndetected $\widehat{=}$
extends  TargetUndetected
    when
        grd1 : $target\_det = FALSE$
        grd2 : $speed < MAX\_SPEED - INC$
        grd3 : $isRain = FALSE$
        grd4 : $isSharp = FALSE$
    then
        act1 : $speed := speed + INC$
    end
Event  RainSharp $\widehat{=}$
    when
        grd1 : $isRain = TRUE \lor isSharp = TRUE$
    then
        act1 : $speed := speed - INC$
    end
END
```

Appendix C

Event-B specifications and proof obligations of Crane Controller Example

This appendix contains full Event-B specification for checking safety and eventuality properties of Crane Controller example in Chapter 5.

C.1 Context specification of Crane Controller system

An Event-B Specification of Crane_C0
Creation Date: 19May2014 @ 09:10:29 AM

CONTEXT Crane_C0

SETS

```
    POWER
    HEDGES
    F_DISTANCE
```

CONSTANTS

    fast
    medium
    zero
    slow
    quite
    very
    start
    far
    close
    above
    precise

AXIOMS

axm1 : partition(POWER, {fast}, {slow}, {zero})

axm2 : partition(HEDGES, {very}, {quite}, {precise})

axm6 : partition(F_DISTANCE, {start}, {far}, {medium}, {close}, {above})

END

C.2 Extended context

An Event-B Specification of Extension

Creation Date: 19May2014 @ 09:10:29 AM

CONTEXT Crane_C1

EXTENDS Crane_C0

CONSTANTS

    deg_DIS
    deg_HED
    deg_POWER

AXIOMS

axm1 : $\mathtt{deg\_POWER} \in \mathtt{POWER} \rightarrow \mathbb{N}$

axm2 : $\mathtt{deg\_DIS} \in \mathtt{F\_DISTANCE} \rightarrow \mathbb{N}$

axm3 : $\mathtt{deg\_HED} \in \mathtt{HEDGES} \rightarrow \mathbb{N}$

axm4 : $\mathtt{deg\_HED}(\mathtt{very}) = 3 \land \mathtt{deg\_HED}(\mathtt{quite}) = 2 \land \mathtt{deg\_HED}(\mathtt{precise}) = 1$

axm5 : $\mathtt{deg\_DIS}(\mathtt{start}) = 4 \land \mathtt{deg\_DIS}(\mathtt{far}) = 3 \land \mathtt{deg\_DIS}(\mathtt{medium}) = 2 \land \mathtt{deg\_DIS}(\mathtt{close}) = 1 \land \mathtt{deg\_DIS}(\mathtt{above}) = 0$

axm6 : $\mathtt{deg\_POWER}(\mathtt{fast}) = 1 \land \mathtt{deg\_POWER}(\mathtt{slow}) = 2 \land \mathtt{deg\_POWER}(\mathtt{zero}) = 3$

END

C.3 Machine specification of Crane Controller system

An Event-B Specification of Crane_M0
Creation Date: 19May2014 @ 09:10:29 AM

MACHINE Crane_M0

SEES Crane_C0

VARIABLES

speed

dist

INVARIANTS

inv2 : $\mathtt{speed} \in \mathbb{P}(\mathtt{HEDGES} \times \mathtt{POWER})$

inv3 : $\mathtt{dist} \in \mathbb{P}(\mathtt{HEDGES} \times \mathtt{F\_DISTANCE})$

inv4 : $\mathrm{ran}(\mathtt{dist}) = \{\mathtt{close}\} \Rightarrow \neg\,(\mathrm{ran}(\mathtt{speed}) = \{\mathtt{fast}\})$

EVENTS

Initialisation

begin

act1 : $\mathtt{speed} := \{\mathtt{precise} \mapsto \mathtt{zero}\}$

```
        act2 : dist := {precise ↦ start}
    end
Event  evt1 ≙
Status anticipated
    when
        grd1 : dist = {precise ↦ start}
    then
        act1 : speed := {precise ↦ fast}
        act2 : dist := {precise ↦ far}
    end
Event  evt2 ≙
Status anticipated
    when
        grd1 : dist = {precise ↦ far}
    then
        act1 : speed := {quite ↦ fast}
        act2 : dist := {precise ↦ medium}
    end
Event  evt3 ≙
Status anticipated
    when
        grd1 : dist = {precise ↦ medium}
    then
        act1 : speed := {precise ↦ slow}
        act2 : dist := {precise ↦ close}
    end
Event  evt4 ≙
Status anticipated
    when
        grd1 : dist = {precise ↦ close}
    then
        act1 : dist := {precise ↦ above}
        act2 : speed := {very ↦ slow}
    end
Event  evt5 ≙
Status anticipated
    when
        grd1 : dist = {precise ↦ above}
    then
        act1 : speed := {precise ↦ zero}
        act2 : dist := {precise ↦ start}
    end
END
```

C.4 Refined machine

An Event-B Specification of Refinement

Creation Date: 19May2014 @ 09:10:29 AM

```
MACHINE  Crane_M1
REFINES  Crane_M0
SEES  Crane_C1
VARIABLES
    dist
    speed
    d
VARIANT
    d
INVARIANTS
    inv1 : d ∈ ℕ
```

DELF : $\mathtt{d} = \mathtt{deg\_DIS}(\mathtt{above}) \Rightarrow \mathtt{d} = \mathtt{deg\_DIS}(\mathtt{start}) \lor \mathtt{d} = \mathtt{deg\_DIS}(\mathtt{far}) \lor \mathtt{d} = \mathtt{deg\_DIS}(\mathtt{medium}) \lor \mathtt{d} = \mathtt{deg\_DIS}(\mathtt{close}) \lor \mathtt{d} = \mathtt{deg\_DIS}(\mathtt{above})$

EVENTS

Initialisation

extended

begin

act1 : $speed := \{precise \mapsto zero\}$

act2 : $dist := \{precise \mapsto start\}$

act3 : $\mathtt{d} := \mathtt{deg\_DIS}(\mathtt{start})$

end

Event *evt1* $\widehat{=}$

Status convergent

extends *evt1*

when

grd1 : $dist = \{precise \mapsto start\}$

grd2 : $\mathtt{d} = \mathtt{deg\_DIS}(\mathtt{start})$

grd3 : $\neg\, \mathtt{d} = \mathtt{deg\_DIS}(\mathtt{above})$

then

act1 : $speed := \{precise \mapsto fast\}$

act2 : $dist := \{precise \mapsto far\}$

act3 : $\mathtt{d} := \mathtt{deg\_DIS}(\mathtt{far})$

end

Event *evt2* $\widehat{=}$

Status convergent

extends *evt2*

when

grd1 : $dist = \{precise \mapsto far\}$

grd2 : $\neg\, \mathtt{d} = \mathtt{deg\_DIS}(\mathtt{above})$

grd3 : $\mathtt{d} = \mathtt{deg\_DIS}(\mathtt{far})$

then

act1 : $speed := \{quite \mapsto fast\}$

act2 : $dist := \{precise \mapsto medium\}$

act3 : $\mathsf{d} := \mathsf{d} - (\mathtt{deg\_DIS}(\mathtt{far}) - \mathtt{deg\_DIS}(\mathtt{medium}))$

end

Event *evt3* $\widehat{=}$

Status convergent

extends *evt3*

when

grd1 : $dist = \{precise \mapsto medium\}$

grd2 : $\neg\ \mathsf{d} = \mathtt{deg\_DIS}(\mathtt{above})$

grd3 : $\mathsf{d} = \mathtt{deg\_DIS}(\mathtt{medium})$

then

act1 : $speed := \{precise \mapsto slow\}$

act2 : $dist := \{precise \mapsto close\}$

act3 : $\mathsf{d} := \mathsf{d} - (\mathtt{deg\_DIS}(\mathtt{medium}) - \mathtt{deg\_DIS}(\mathtt{close}))$

end

Event *evt4* $\widehat{=}$

Status convergent

extends *evt4*

when

grd1 : $dist = \{precise \mapsto close\}$

grd2 : $\neg\ \mathsf{d} = \mathtt{deg\_DIS}(\mathtt{above})$

grd3 : $\mathsf{d} = \mathtt{deg\_DIS}(\mathtt{close})$

then

act1 : $dist := \{precise \mapsto above\}$

act2 : $speed := \{very \mapsto slow\}$

act3 : $\mathsf{d} := \mathsf{d} - (\mathtt{deg\_DIS}(\mathtt{close}) - \mathtt{deg\_DIS}(\mathtt{above}))$

end

Event *evt5* $\widehat{=}$

Status convergent

extends *evt5*

when

grd1 : $dist = \{precise \mapsto above\}$

grd2 : $\neg$ d = deg_DIS(above)

grd3 : d = deg_DIS(above)

then

act1 : $speed := \{precise \mapsto zero\}$

act2 : $dist := \{precise \mapsto start\}$

act3 : d := d − (deg_DIS(above) − deg_DIS(start))

end

END

C.5 Proof obligations for checking the safety property

In this section, we list all proof obligations of each event in machine $Crane\_M0$ that need to be proved to show the correctness of safety properties.

TABLE C.1: INV PO of event $evt1$

$ran(dis) = \{close\} \Rightarrow \neg ran(speed) = \{fast\}$ $dis = \{precise \mapsto start\}$ $\vdash$ $ran(\{precise \mapsto far\}) = \{close\} \Rightarrow \neg ran(\{precise \mapsto fast\}) = \{fast\}$	$evt1/inv4/INV$

TABLE C.2: INV PO of event $evt2$

$ran(dis) = \{close\} \Rightarrow \neg ran(speed) = \{fast\}$ $dis = \{precise \mapsto far\}$ $\vdash$ $ran(\{precise \mapsto medium\}) = \{close\} \Rightarrow \neg ran(\{quite \mapsto fast\}) = \{fast\}$	$evt2/inv4/INV$

TABLE C.3: INV PO of event $evt3$

$ran(dis) = \{close\} \Rightarrow \neg ran(speed) = \{fast\}$ $dis = \{precise \mapsto medium\}$ $\vdash$ $ran(\{precise \mapsto close\}) = \{close\} \Rightarrow \neg ran(\{precise \mapsto slow\}) = \{fast\}$	$evt3/inv4/INV$

TABLE C.4: INV PO of event *evt5*

$ran(dis) = \{close\} \Rightarrow \neg ran(speed) = \{fast\}$ $dis = \{precise \mapsto above\}$ $\vdash$ $ran(\{precise \mapsto start\}) = \{close\} \Rightarrow \neg ran(\{precise \mapsto zero\}) = \{fast\}$	$evt5/inv4/INV$

C.6 Proof obligations for checking convergence properties

In this section, we list all proof obligations of each convergent event in machine *Crane_M*1 that need to be proved to show the variant decreases after its execution (*VARPO*) and has type of Natural number (*NATPO*).

TABLE C.5: VAR PO of event *evt1*

$dis = \{precise \mapsto start\}$ $d = deg\_DIS(start)$ $\neg d = deg\_DIS(above)$ $\vdash$ $deg\_DIS(far) < d$	evt1/VAR

TABLE C.6: NAT PO of event *evt1*

$deg\_DIS \in F\_DISTANCE \rightarrow N$ $dis = \{precise \mapsto start\}$ $d = deg\_DIS(start)$ $\neg d = deg\_DIS(above)$ $\vdash$ $d \in \mathbb{N}$	evt1/NAT

TABLE C.7: VAR PO of event *evt2*

$dis = \{precise \mapsto far\}$ $d = deg\_DIS(far)$ $\neg d = deg\_DIS(above)$ $\vdash$ $d - (deg\_DIS(far) - deg\_DIS(medium)) < d$	evt2/VAR

Table C.8: NAT PO of event *evt*2

$deg\_DIS \in F\_DISTANCE \to N$ $dis = \{precise \mapsto far\}$ $d = deg\_DIS(far)$ $\neg d = deg\_DIS(above)$ $\vdash$ $\vdash$ $d \in \mathbb{N}$	evt2/NAT

Table C.9: VAR PO of event *evt*3

$dis = \{precise \mapsto medium\}$ $\neg d = deg\_DIS(close)$ $d = deg\_DIS(medium)$ $\vdash$ $d - (deg\_DIS(medium) - deg\_DIS(close)) < d$	evt3/VAR

Table C.10: NAT PO of event *evt*3

$deg\_DIS \in F\_DISTANCE \to N$ $dis = \{precise \mapsto medium\}$ $\neg d = deg\_DIS(close)$ $d = deg\_DIS(medium)$ $\vdash$ $\vdash$ $d \in \mathbb{N}$	evt3/NAT

Table C.11: VAR PO of event *evt*5

$dis = \{precise \mapsto above\}$ $\neg d = deg\_DIS(above)$ $d = deg\_DIS(above)$ $\vdash$ $d - (deg\_DIS(above) - deg\_DIS(start)) < d$	evt5/VAR

Table C.12: NAT PO of event *evt*5

$deg\_DIS \in F\_DISTANCE \to N$ $dis = \{precise \mapsto above\}$ $\neg d = deg\_DIS(above)$ $d = deg\_DIS(above)$ $\vdash$ $\vdash$ $d \in \mathbb{N}$	evt5/NAT

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

LÊ HỒNG ANH

METHODS FOR MODELING AND VERIFYING EVENT-DRIVEN SYSTEMS

DOTORAL THESIS IN INFORMATION TECHNOLOGY

Hà Nội – 2015

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BẢNG KÊ NHẬN TIỀN

Nội dung: Thù lao cho Tiểu ban đánh giá hồ sơ chuyên môn của thí sinh đào tạo tiến sĩ đợt 1/2015 chuyên ngành Kỹ thuật Viễn thông

STT	Họ và tên	Đơn vị	Số tiền	Ký nhận
1	Nguyễn Quốc Tuấn	Trưởng tiểu ban	200.000	[illegible]
2	Đinh Triều Dương	Thư ký	150.000	[illegible]
3	Chử Đức Trình	Uỷ viên	100.000	[illegible]
4	Đặng Thế Ngọc	Uỷ viên	100.000	[illegible]
5	Nguyễn Nam Hoàng	Uỷ viên	100.000	[illegible]
	Cộng:		650.000	

Tổng số tiền bằng chữ: Sáu trăm năm mươi nghìn đồng chẵn./

Hà Nội, ngày 22 tháng 4 năm 2015

NGƯỜI DUYỆT	PHỤ TRÁCH ĐƠN VỊ	NGƯỜI LẬP BIỂU
		[illegible]
		Lại Thanh Hùng

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BẢNG KÊ NHẬN TIỀN

Nội dung chi: Thù lao tiểu ban đánh giá hồ sơ chuyên môn

STT	Họ và tên	Đơn vị	Số tiền	Ký nhận
1	GS Nguyễn Năng Định	Khoa VLKT & CNNN	200.000	[illegible]
2	TS Đặng Đình Long	Khoa VLKT & CNNN	150.000	[illegible]
3	PGS Hoàng Nam Nhật	Khoa VLKT & CNNN	100.000	[illegible]
4	PGS Đỗ T. Hương Giang	Khoa VLKT & CNNN	100.000	[illegible]
5	TS Đỗ Hoàng Tùng	Viện Vật lý	100.000	[illegible]
	Cộng:		650.000đ	

Tổng số tiền bằng chữ: Sáu trăm năm mươi ngàn đồng

Hà Nội, ngày 15 tháng 4 năm 2015

NGƯỜI DUYỆT	PHỤ TRÁCH ĐƠN VỊ	NGƯỜI LẬP BIỂU
		[illegible]
		Nguyễn Thị Hằng

BẢNG KÊ CHỨNG TỪ CHI MỤC: Nhận xét của thành viên tiểu ban về hồ sơ chuyên môn

1.	Nhận xét của tiểu ban chuyên ngành Kỹ thuật phần mềm..	.... 500.000 đ
2.	Nhận xét của tiểu ban chuyên ngành Hệ thống thông tin ...	.... 1.500.000 đ
3.	Nhận xét của tiểu ban chuyên ngành Kỹ thuật viễn thông..	.... 500.000 đ
4.	Nhận xét của tiểu ban chuyên ngành Vật liệu và linh kiện nano	.... 1.500.000 đ
		Số tiền .... **4.000.000 đ**

(Viết bằng chữ: *Bốn triệu đồng chẵn)*

Kèm theo chứng từ gốc

Phụ trách đơn vị	**Người đề nghị**
[illegible]	[illegible]
Nguyễn Phương Thái	**Dương Đình Thiệu**

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BẢNG KÊ NHẬN TIỀN

Nội dung chi: Thù lao đọc hồ sơ chuyên môn của thí sinh dự thi Tiến sĩ, chuyên ngành Kỹ thuật Phần mềm

STT	Họ và tên	Đơn vị	Chức trách HĐ	Số tiền	Ký nhận
1	PGS.TS. Trương Anh Hoàng	Khoa CNTT, Trường ĐHCN	Trưởng tiểu ban	100,000	[illegible]
2	TS. Tô Văn Khánh	Khoa CNTT, Trường ĐHCN	Ủy viên thư ký	100,000	[illegible]
3	PGS.TS. Trương Ninh Thuận	Khoa CNTT, Trường ĐHCN	Ủy viên	100,000	[illegible]
4	TS. Phạm Ngọc Hùng	Khoa CNTT, Trường ĐHCN	Ủy viên	100,000	[illegible]
5	TS. Đặng Văn Hưng	Khoa CNTT, Trường ĐHCN	Ủy viên	100,000	[illegible]
6					
7					
8					
9					
10					
TỔNG CỘNG				500.000	

Bằng chữ: Năm trăm nghìn đồng chẵn

Hà Nội, ngày 16 tháng 4 năm 2015

NGƯỜI DUYỆT	**PHỤ TRÁCH ĐƠN VỊ**	**NGƯỜI LẬP**
	[illegible]	[illegible]
	Trương Ninh Thuận	**Mạnh Phương Anh**

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BẢNG KÊ NHẬN TIỀN

Nội dung chi: Thù lao đọc hồ sơ chuyên môn của thí sinh dự thi Tiến sĩ, chuyên ngành Hệ thống thông tin

STT	Họ và tên	Đơn vị	Chức trách HĐ	Số tiền	Ký nhận
1	TS. Nguyễn Ngọc Hóa	Khoa CNTT, Trường ĐHCN	Trưởng tiểu ban	300,000	[illegible]
2	PGS.TS. Nguyễn Trí Thành	Khoa CNTT, Trường ĐHCN	Ủy viên thư ký	300,000	[illegible]
3	PGS.TS. Nguyễn Hải Châu	Khoa CNTT, Trường ĐHCN	Ủy viên	300,000	[illegible]
4	PGS.TS. Nguyễn Hà Nam	Khoa CNTT, Trường ĐHCN	Ủy viên	300,000	[illegible]
5	PGS.TS. Hà Quang Thụy	Khoa CNTT, Trường ĐHCN	Ủy viên	300,000	[illegible]
6					
7					
8					
9					
10					
TỔNG CỘNG				1.500,000	

Bằng chữ: Một triệu năm trăm nghìn đồng chẵn.

Hà Nội, ngày 22 tháng 4 năm 2015

NGƯỜI DUYỆT	PHỤ TRÁCH ĐƠN VỊ	NGƯỜI LẬP
	[illegible]	[illegible]
	Trương Ninh Thuận	Mạnh Phương Anh

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BẢNG KÊ NHẬN TIỀN

Nội dung: Nhận xét của các thành viên Tiểu ban đánh giá hồ sơ chuyên môn của thí sinh đào tạo tiến sĩ đợt 1/2015 chuyên ngành Kỹ thuật Viễn thông

STT	Họ và tên	Đơn vị	Số tiền	Ký nhận
1	Nguyễn Quốc Tuấn	Trưởng tiểu ban	100.000	[illegible]
2	Đinh Triều Dương	Thư ký	100.000	[illegible]
3	Chử Đức Trình	Uỷ viên	100.000	[illegible]
4	Đặng Thế Ngọc	Uỷ viên	100.000	[illegible]
5	Nguyễn Nam Hoàng	Uỷ viên	100.000	[illegible]
	Cộng:		500.000	

Tổng số tiền bằng chữ: Năm trăm nghìn đồng chẵn./

Hà Nội, ngày 22 tháng 04 năm 2015

NGƯỜI DUYỆT	PHỤ TRÁCH ĐƠN VỊ	NGƯỜI LẬP BIỂU
	[illegible]	[illegible]
		Lại Thanh Hương

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BẢNG KÊ NHẬN TIỀN

Nội dung chi: Thù lao nhận xét của tiểu ban đánh giá hồ sơ chuyên môn.

STT	Họ và tên	Đơn vị	Số tiền	Ký nhận
1	GS Nguyễn Năng Định	Khoa VLKT & CNNN	300.000	[illegible]
2	TS Đặng Đình Long	Khoa VLKT & CNNN	300.000	[illegible]
3	PGS Hoàng Nam Nhật	Khoa VLKT & CNNN	300.000	[illegible]
4	PGS Đỗ T Hương Giang	Trường ĐHCN, khoa	300.000	[illegible]
5	TS Đỗ Hoàng Tùng	Viện Vật lý	300.000	[illegible]
	Cộng:		1.500.000 đ	

Tổng số tiền bằng chữ: Một triệu năm trăm ngàn đồng

Hà Nội, ngày 15 tháng 4 năm 2015

NGƯỜI DUYỆT	KT. PHỤ TRÁCH ĐƠN VỊ	NGƯỜI LẬP BIỂU
	[illegible]	[illegible]
		Nguyễn Thị Hạnh

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

---------- o 0 o ----------

GIẤY BIÊN NHẬN

Tên tôi là: Dương Đình Thiều

Địa chỉ: Phòng Đào tạo

Điện thoại:

Tôi đã nhận của:

Số tiền: 400.000

(Bằng chữ: Bốn trăm nghìn đồng chẵn *)*

Lý do: Chuẩn bị tài liệu cho các TB cuộc họp đánh giá HSCM (04 TB x 100.000đ/TB).

Hà Nội, ngày 22 tháng 4 năm 2015

Xác nhận chi	**Người giao tiền**	**Người nhận tiền**
[illegible]		[illegible]
		D.Đ. Thiều

BẢNG KÊ CHỨNG TỪ CHI MỤC: Phục vụ Tiểu ban chuyên môn sinh đào tạo tiến sĩ đợt 1 năm 2015 họp

1.	Phục vụ các chuyên ngành KTPM, HTTT	................150.000 đ
2.	Phục vụ chuyên ngành Kỹ thuật viễn thông........................	.................75.000 đ
3.	Phục vụ chuyên ngành Vật liệu và linh kiện nano.............	.................75.000 đ
4.	Phục vụ các chuyên ngành	................300.000 đ
		Số tiền ....**600.000 đ**

(Viết bằng chữ: *Sáu trăm nghìn đồng chẵn)*

Kèm theo chứng từ gốc

Phụ trách đơn vị **Người đề nghị**

[illegible] [illegible]

Nguyễn Phương Thái **Dương Đình Thiệu**

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

--------- o 0 o ---------

GIẤY BIÊN NHẬN

Tên tôi là: Mạnh Phương Anh

Địa chỉ: Khoa Công nghệ thông tin, Trường Đại học Công nghệ

Điện thoại: 0974.206.722

Tôi đã nhận của: ....

Số tiền: 150.000 đ

(Bằng chữ: Một trăm năm mươi nghìn đồng chẵn *)*

Lý do: Phục vụ tiểu ban đánh giá hồ sơ chuyên môn

Hà Nội, ngày 22 tháng 4 năm 2015

Xác nhận chi	**Người giao tiền**	**Người nhận tiền**
[illegible]	[illegible] D.A. Thiệu	[illegible] Mạnh Phương Anh

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

--------- o 0 o ---------

GIẤY BIÊN NHẬN

Tên tôi là: Nguyễn Thị Hạnh

Địa chỉ: Khoa VLKT & CNNN

Điện thoại: (04) 3759 9429

Tôi đã nhận của: CV Dương Anh Thiệu

Số tiền: 75.000 đ

(Bằng chữ: Bảy mươi lăm nghìn đồng chẵn *)*

Lý do: Phục vụ Tiểu ban đánh giá HS cán chuyên ngành Vật lý kỹ thuật và linh kiện nano họp xét thi NCS đợt 11.2015

Hà Nội, ngày 15 tháng 9 năm 2015

Xác nhận chi	**Người giao tiền**	**Người nhận tiền**
[illegible]	[illegible] D.A. Thiệu	[illegible] Nguyễn Thị Hạnh

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

---------- o0o ----------

GIẤY BIÊN NHẬN

Tên tôi là: Lại Thanh Hương

Địa chỉ: Khoa Điện tử - Viễn thông

Điện thoại: (04) 3754 9338

Tôi đã nhận của: CV Dương Đình Thiều

Số tiền: 75.000đ

(Bằng chữ: Bảy mươi lăm nghìn đồng chẵn)

Lý do: Phục vụ TBCM đánh giá HSMT họp

Hà Nội, ngày 22 *tháng* 4 *năm 20*15

Xác nhận chi	**Người giao tiền**	**Người nhận tiền**
[illegible]	[illegible] D.Đ. Thiều	[illegible] Lại Thanh Hương

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

---------- o0o ----------

GIẤY BIÊN NHẬN

Tên tôi là: Dương Đình Thiều

Địa chỉ: Phòng Đào tạo

Điện thoại:

Tôi đã nhận của:

Số tiền: 300.000

(Bằng chữ: Ba trăm nghìn đồng chẵn)

Lý do: Phục vụ các TBCM đánh giá HSMT đợt 1/2015 họp

Hà Nội, ngày 22 *tháng* 4 *năm 20*15

Xác nhận chi	**Người giao tiền**	**Người nhận tiền**
[illegible]		[illegible] Dương Đình Thiều