Le Hong Anh

Methods for modeling and verifying event-driven systems

SUMMARY OF DOCTORAL THESIS IN INFORMATION TECHNOLOGY
Major: Software Engineering
Code: 62.48.10.01

Ha Noi, 2014
This work was done at Department of Software Engineering, Faculty of Information Technology, VNU - University of Engineering and Technology.

Supervisors:
Assoc. Prof. Dr. Truong Ninh Thuan
Assoc. Prof. Dr. Pham Bao Son

Reviewer 1: ....................
Reviewer 2: ....................
Reviewer 3: ....................

This thesis can be found at
- National Library of Vietnam
- Library and Information Center - VNU Hanoi
Chapter 1. Introduction

1.1 Motivation

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. Testing techniques can be used in normal development in order to check whether the software execution satisfies users requirements. However, testing is always an incomplete validation because it can only identifies errors in some cases 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, theorem-proving and program analysis. 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. On the other hand, software architecture is a concept proposed a way to effectively build complex software systems. A typical type of software architecture or design styles usually has several suitable modeling and verification methods.

Event-driven architecture is one of the most popular architectures in software project development providing implicit invocation instead of invoking routines directly such that each component can produce events, the system then invoke all procedures registered with these events. 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 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 [11]. In event-driven architectures, 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. The existing methods for modeling and verification of general event-driven systems are insufficient because we often develop particular types of event-driven systems which use ECA rules to react to raise events, e.g., active databases and context-aware systems. Furthermore, almost existing work of software verification...
focuses on analysing precise descriptions of required functionality and behavior of the system. For these reasons, new methods or approaches to 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 development. It is also beneficial if it reduces the complexity of proving and is practical in software development. The thesis proposes novel methods to achieve that desire by using Event-B formal method. 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. Hence, Event-B is totally matched for modeling and verifying event-driven systems.

1.2 Objectives
The thesis aims to provide a different approach in comparison with 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 proposes effective methods which 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 has another objective to analyse event-driven systems whose behavior is described by imprecise requirements (represented by Fuzzy If-Then rules). The thesis introduces a new refinement-based method for modeling and verifying both safety and eventuality properties of such systems.

1.3 Contributions
Research contributions of this thesis are as follows

1. This thesis introduces a new method to model and verify a database system with triggers by 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. With the proposed method, constraint preservation properties are verified and infinite loops are detected by formal proofs. 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 systems with triggers is also developed.
2. The thesis continues investigating the benefit of similar acts between ECA rules and Event-B event to propose a method to model and verify context-aware systems. Furthermore, the thesis recognizes the advantages of Event-B refinement mechanism to make proposed methods suitable for incremental modeling. Significant properties such as context constraint preservation are defined as invariants and can be checked automatically using the supporting tool Rodin.

3. We handle the case that a system is described by imprecise requirements. Its behavior rules are now specified in the form of Fuzzy If-Then rules. The thesis introduce 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.

4. The thesis makes use of Event-B refinement and some existing reasoning methods to analyse some significant properties of imprecise system requirements such as safety and eventuality properties.

1.4 Thesis structure

The remainder of this thesis is organized as follows:

Chapter 2 provides necessary backgrounds for the thesis. Chapter 3 introduces a new method for modeling and verifying database systems. Chapter 4 focuses on modeling and verifying context-aware systems. In Chapter 5, we aa modeling method for the case that an event-driven system is described by Fuzzy If-Then rules. Chapter 6 presents a new method which makes use of Event-B refinement and existing methods to verify safety and eventuality properties of the system. Chapter 7 summarizes the
thesis and present future work.

Chapter 2. Backgrounds

2.1 Temporal logic

Propositional temporal logic (PTL) extends the descriptive power of propositional logic to describe a sequence of states in different moment of time called 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 $\land$, $\lor$, $\neg$; and the “temporal” operators $\Box$ (“always”), $\Diamond$ (“eventually”), and $\circ$ (“next”), $U$ (“until”). The syntax of a PTL formula is defined in Table 2.1

<table>
<thead>
<tr>
<th>Syntax of PTL formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle formula \rangle := \langle term \rangle \lor \langle term \rangle$</td>
</tr>
<tr>
<td>$\langle term \rangle &gt; := \langle factor \rangle \land \langle factor \rangle$</td>
</tr>
<tr>
<td>$\langle factor \rangle &gt; := \langle primary \rangle {U \langle primary \rangle}$</td>
</tr>
<tr>
<td>$\langle primary \rangle &gt; := \langle atomic - proposition \rangle &gt;$</td>
</tr>
<tr>
<td>$\neg \langle primary \rangle &gt;$</td>
</tr>
<tr>
<td>$\circ \langle primary \rangle &gt;$</td>
</tr>
<tr>
<td>$\Diamond \langle primary \rangle &gt;$</td>
</tr>
<tr>
<td>$\Box \langle primary \rangle &gt;$</td>
</tr>
</tbody>
</table>

2.2 Classical set theory

Sets are fundamental objects 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 $B$ ($A \in B$), it means that $B$ contains $A$ as an element [14]. There are some basic definitions of set theory such as power set, relations, functions, lamda notation, etc..

2.3 Fuzzy sets and Fuzzy If-Then rules

Many real-world software systems are developed from requirements of all stakeholders. In fact, stakeholders usually can not describe the system precisely. They often use vague, ambiguous, fuzzy terms such as “very good”, “far”, “hot”, etc.. In order to deal with systems which are too complex or too ill-defined to admit of precise descriptions, Zadeh [16] introduced a logic framework which is not traditional two-valued, but multi-valued logics whose values are interpreted by Fuzzy sets. A fuzzy set $F$ defined on an universal set $X$ is a set, each element of which is a pair of values as follows:

$$A = \{(x,) \mu_A(x)\}, x \in X$$  \hspace{1cm} (2.1)

where $\mu_A(x) : X \rightarrow [0,1]$ is termed as the grade of membership of $x$ in $A$.

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 $not\ very\ tall$. Fuzzy If-Then rules, written in a simple form: “If $a$ is $A$ then $b$ is $B$”, play an important role in fuzzy sets. It
Chapter 2 Backgrounds

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

Formal methods which can be used to specify and verify systems mathematically. A method is formal if it has well-defined mathematics basis, typically given by a formal specification language [13]. 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. 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 [8], Z [12], B [2].

2.4.1 VDM

VDM stands for “The Vienna Development Method” which is 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 [8]. 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 model-oriented specification language called VDM-SL. It 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).

2.4.2 Z method

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 problem, 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 [12].

2.4.3 B method

B is a 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 [2]. 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 modules which can be viewed or used in other modules.

2.5 Event-B

Event-B [3] is a formal method for system-level modeling and analysis. 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. A basic structure of an Event-B model consists of MACHINE and CONTEXT.

An Event B CONTEXT describes a static part where all the relevant properties and hypotheses are defined. A CONTEXT consists of carrier sets, constants, axioms. Carrier sets, denoted by $s$, are represented by their names, and are non-empty. Different carrier sets are completely independent. The constants $c$ are defined by means of a number of axioms $P(s, c)$ also depending on the carrier sets $s$.

A MACHINE is defined by a set of clauses. A machine is composed of variables, invariants, theorems and events. Variables $v$ are representing states of the model. Invariants $I(v)$ yield the laws that state variables $v$ must always be satisfied. These laws are formalized by means of predicates expressed within the language of First Order Predicate Calculus with Equality extended by Set Theory. Events $E(v)$ present transitions between states. Each event has the form $evt = \text{any } x$ where $G(x, v)$ then $A(x, v, v')$ end where $x$ are local variables of the event, $G(x, v)$ is a guard condition and $A(x, v, v')$ is an action. An event is enabled when its guard condition is satisfied. The event action consists of one or more assignments. We have three kinds of assignments for expressing the actions associated with an event: (1) a deterministic multiple assignment $(x := E(t, v))$, (2) an empty assignment (skip), or (3) a non-deterministic multiple assignment $(x :| P(t, v, x'))$. 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 model. A concrete Event-B machine can refine at most one abstract machine. A refined machine usually has more variables than its abstraction as we have new variables to represent more details of the model. In superposition refinement, the abstract variables are retained in the concrete machine, with possibly some additional variables. In vertical refinement such as data refinement, 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 $J(v, w)$. In order to check if a machine satisfies a collection of specified properties, Event-B defines proof obligations (POs) which we must prove. Some of the proof obligations relevant to thesis are invariant preservation (INV), convergence (VAR), deadlock-freeness (DLF).

### 2.6 Rodin tool

This thesis uses the RODIN toolkit version 2.8 [1] which is an Eclipse environment for modeling and proving in Event-B. It is built on top the Eclipse platform and is a set of plug-ins containing tools used to support modeling and a graphical user interface. The Rodin tool provides a rich of perspective windows to user such as proving, Event-B editors, etc.. 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
There are many types of event-driven systems including software user interfaces, 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 [11]. 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.

2.7.1 Database systems and database triggers
A relational database system which is based on the relational model consists of collections of objects and relations, operations for manipulation and data integrity for accuracy and consistency. Modern relational database systems include active rules as database triggers which response 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. Database triggers can be mainly classified by two kind: 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..

2.7.2 Context-aware systems
The term “context-aware” was first introduced by Bill Schilit [10], 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. In this thesis, we focus on a context-aware system which directly use 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.

Chapter 3. Modeling and verifying database systems

3.1 Introduction
A trigger is made of a block of code and has a syntax. It is 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 are desirable. Moreover, it is valuable if we can show that triggers execution is correct at the design time because it reduces the cost of database application 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 makes use of Event-B proof obligations to prove some 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. Furthermore, the method is such practical that we can implement a tool following the main idea to transform a database model to an Event-B model in Rodin platform automatically (or partly). It also overcomes one of disadvantages that makes formal methods absent in the database development process because of the modeling complexity.

3.2 Modeling and verifying database triggers system

3.2.1 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. The translation rules are summarized in Table 3.1.

<table>
<thead>
<tr>
<th>Database definitions</th>
<th>Event-B concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1. $db = \langle T, C, G \rangle$</td>
<td>$DB_M, DB_C$</td>
</tr>
<tr>
<td>Rule 2. $t = \langle r_1, \ldots, r_m \rangle$</td>
<td>$T = TYPE_1 \times TYPE_2 \times TYPE_n$</td>
</tr>
<tr>
<td>Rule 3. $r_i = \langle f_{i1}, \ldots, f_{in} \rangle$</td>
<td>$t \in T$</td>
</tr>
<tr>
<td>Rule 4. Primary key constraint</td>
<td>$f : TYPE_1 \rightarrow T$</td>
</tr>
<tr>
<td>Rule 5. Constraint $C$</td>
<td>Invariant $I$</td>
</tr>
<tr>
<td>Rule 6. Trigger $E$</td>
<td>Event $Evt$</td>
</tr>
</tbody>
</table>

3.2.2 Formalizing triggers

As illustrated in Table 3.2, a trigger is translated to an Event-B event where conjunction of trigger’s type and its condition is the guard of the event. The action of the trigger is translated to the body part of an Event-B event. We assume that it contains a single DML statement such as delete, insert, update. The encoding of the trigger action is illustrated in Table 3.3
### 3.2.3 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 infinite loop. This situation occurs when after a sequence of events, state of the system does not change. With the proposed method, there are two ways to check this property of the system. The first one use 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

\[ I(v), G(c) \models G_1(v) \lor ... \lor G_n(v), \]

where \( v \) is variable, \( I(v) \) denotes invariant, \( G_i(v) \) presents guard of the event. 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 \textit{INITIALISATION} event.

- **Constraint preservation:** With the proposed translation method, a trigger does not break these rules if \( I(v), G(w, v), S(w, v, v') \models I(v') \). It means that data constraints are preserved after executing the trigger. This is also the INV proof obligation of Event-B events.

### 3.3 An example

#### 3.3.1 Example description

Let assume that we have a database system including two tables \texttt{EMPLOYEES} which has two columns of employee identifier and level, while table \texttt{BONUS} contains two columns of employee identifier and amount. The database system has a constraint: \textit{The bonus of an employee with a level greater than 5 is at least 10}. 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 with amount that is greater than 10, then his level is increased by 1.

3.3.2 Modeling the example
The Event-B specification of the example is partly shown in Figure 3.1, Figure 3.2, Figure 3.3.

```plaintext
CONTEXT TRIGGER_C
SETS
  TYPES
  TABLE_NAMES
CONSTANTS
  TBL_EMPL
  TBL_BONUS
AXIOMS
  axm1: partition(TYPES, {insert}, {update}, {delete})
  axm2: TBL_EMPL ⊆ {1→N, 2→N}
  axm3: TBL_BONUS ⊆ {1→N, 2→N}
  axm4: partition(TABLE_NAMES, {employees}, {bonus})
END
```

**Figure 3.1: A part of Event-B Context**

```plaintext
MACHINE TRIGGER_M
SEES TRIGGER_C
VARIABLES
  bonus
  empl
  f_bonus
  f_empl
  type
INVARIANTS
  inv1: bonus ∈ P(TBL_BONUS)
  inv2: empl ∈ P(TBL_EMPL)
  inv3: type ∈ TYPES
  inv4: f_bonus ∈ N ⟷ N
  inv5: f_empl ∈ N ⟷ N
  SYS_CTR: ∀ eid.eid ∈ dom(empl) ∧ pk_empl(eid) > 5 ⟷ pk_bonus(eid) > 10
  INF_LOOP: (type = update ∧ table = BONUS) ∨ (type = update ∧ table = EMPL)
END
```

**Figure 3.2: A part of Event-B machine**

3.3.3 Checking properties
- Constraint preservation: Since the constraint property of the system is modeled by the invariant
  \( SYS_CTR: \forall \text{eid.eid} \in \text{dom}(\text{empl}) \land \text{pk}_\text{empl}(\text{eid}) > 5 \Rightarrow \text{pk}_\text{bonus}(\text{eid}) > 10 \).
  We need to prove that the invariant is maintained before and after events execution. The proof obligation of `trigger1` is illustrated in Table 3.4. Two events `Trigger1` and `Trigger2` of the machine `DB_M` generate two proof obligations called `trigger1/SYS_CTR/INV`, `trigger2/SYS_CTR/INV` respectively.
- Infinite loop: In Section 3.2.3, we proposed that a invariant \( INF\_LOOP \) which is the disjunction of the event guards is added to the target machine. If we show
Chapter 3 Modeling and verifying database systems

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 3.3: Encoding trigger

Table 3.4: INV PO of event trigger1.

| ∀ nid. nid ∈ dom(empl_rec) ∧ pk_empl(nid) > 5 ⇒ pk_bonus(nid) > 10
  emplid ∈ dom(empl_rec)
type = update
table = EMPL
| trigger1/SYS_CTR/
| INV

that this invariant is preserved by machine DB_M0, then two triggers execution leads to the infinite loop. The proof clause of the event trigger1 is presented in Table 3.5.

Table 3.5: INV PO of event trigger1

| ∀ nid. (nid ∈ dom(empl_rec) ∧
  type = update ∧ table = BONUS ∧
  pk_bonus(nid) ≥ 10) ∨ (type = update ∧ table = EMPL)) ∧
  emplid ∈ dom(bonus_rec)
table = BONUS ∧ pk_bonus(emplid) ≥ 10
| trigger1/INF_LOOP/
| INV

3.4 Implementation

Following the method presented in Section 3.2, we implement a tool called Trigger2B to support designing and modeling a database system including trigger. 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 3.4.

![Figure 3.4: Architecture of Trigger2B tool](image)

**Chapter 4. Modeling and verification of context-aware systems**

**4.1 Introduction**

Context awareness of an application relates to adaptation, responsiveness, sensitivity of the application to changes of the context [5]. 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 responses to these changes with the providing context knowledge. The behavior of context-ware systems is often complex and uncertain. 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. 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.

**4.2 Formalizing context awareness**

**4.2.1 Modeling context-aware system**

Translation rules between a context-aware system and an Event-B model are presented in Table 4.1

**4.2.2 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...
Table 4.1: Transformation between context-aware systems and Event-B notations

<table>
<thead>
<tr>
<th>Rule</th>
<th>Context-aware concepts</th>
<th>Event-B notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1</td>
<td>Context data ( CD )</td>
<td>Sets, Constants</td>
</tr>
<tr>
<td>Rule 2</td>
<td>Context rules ( r = (e, c, a) )</td>
<td>Events</td>
</tr>
<tr>
<td>Rule 3</td>
<td>Environments triggers ( E )</td>
<td>Events</td>
</tr>
<tr>
<td>Rule 4</td>
<td>Context constraints ( CC )</td>
<td>Invariants</td>
</tr>
</tbody>
</table>

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. Figure 4.1 depicts a incremental modeling method which is based on the proposed method in Section 4.2.1.

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.

- Static part: when a new sensor is added to the system, we may have to deal with new context data types. 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. 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.

4.3 A case study: Adaptive Cruise Control system

4.3.1 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 and is initially set to a value. If the car does not detect a target then ACC increase the speed, other wise
decrease speed with constant amount. If the car is stopped and 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.

4.3.2 Modeling ACC system

In this scenario, there are three sensors, following the approach presented in Section 4.2, we specify the initial system with one abstract machine and one context, namely ACC_M0 and Target (Figure 4.2).

```
EVENTS
Event TargetDetected \equiv
   when
grd1: target_det = TRUE
   grd2: speed > INC
   then
   act1: speed := speed - INC
   end

Event TargetUndetected \equiv
   when
grd1: target_det = FALSE
   grd2: speed < MAX_SPEED - INC
   then
   act1: speed := speed + INC
   end

END
```

Figure 4.2: Events with strengthened guards

4.3.3 Refinement: Adding weather and road sensors

Weather and road sensors are attached to the system. Similarly to target detection sensor, they send 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 speed can not be equal to initial speed if it is raining or the road is sharp”.

Refined model: Following the method presented 4.2.2, context Weather_Road extending context Target represents context data of new sensors. We add two events 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 (Figure 4.3).

4.3.4 Verifying the system’s properties

Context constraints are translated to invariant clauses. Consequently, we prove the system’s correctness by proving proof obligations of such invariants. The proof obligations (PO) for these invariants of both abstract and refined machines as follows:
Figure 4.3: Refined Event-B model for ACC system

Table 4.2: Proof of context constraint preservation

<table>
<thead>
<tr>
<th>Condition</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>( target_det = \text{TRUE} \Rightarrow \text{speed} &lt; \text{MAX_SPEED} )</td>
<td>TargetDetected/ctx_ct1/INV</td>
</tr>
<tr>
<td>( target_det = \text{TRUE} )</td>
<td></td>
</tr>
<tr>
<td>( \text{speed} &gt; \text{INC} )</td>
<td></td>
</tr>
<tr>
<td>( \neg )</td>
<td></td>
</tr>
<tr>
<td>( target_det = \text{TRUE} \Rightarrow \text{speed} - \text{INC} &lt; \text{MAX_SPEED} )</td>
<td></td>
</tr>
</tbody>
</table>

- Machine \( ACC\_M0 \): “TargetDetected/ctx_ct1/INV” (Figure 4.2) and “TargetUndetected/ctx_ct1/INV”
- Machine \( ACC\_M1 \): “TargetUndetected/ctx_ct/INV” and “RainSharp/ctx_ct/INV”

Chapter 5. Modeling 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. 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 [16], is one such formal framework, where the Fuzzy If-Then rules are sometimes employed to represent imprecise system requirements. 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 [7, 15] or Z notation [9].

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.

5.2 Event-B for fuzzy requirements modeling and verification

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.

5.2.1 Representation of fuzzy terms in classical sets

**Corollary 5.1.** A collection of well-defined fuzzy requirements can be specified by classical sets.

**Proof.** Suppose that, fuzzy requirements of a system are specified by \( FR = \{ FR_i \} \), \( FR_i = \{ x_i, m_i, \delta_i, \mu_s, Y_i, P_i \} \), \( i = 1, n \). 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 \( \delta_i \) 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, \( \delta \) and \( Y \), respectively. According to the Definition 4 and 5, \( \delta_i Y_i \) is a membership of the Cartesian product of 2 sets \( \delta \times Y \), \( \mu_s \) is a partial function from Natural number to \( \delta \). Consequently, every elements in \( FR \) can be specified by classical sets. □

5.2.2 Modeling discrete states of imprecise requirements

A system consisting a collection of requirements \( FR_i, i = 1, n \) is modeled by an Event-B model \( FR_B = \{ FR_C, FR_M \} \), 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 translation process is that we can preserve the structure and represent all fuzzy requirements using the Event-B notation. **Translation rules:**

1. **Translation of fuzzy requirements to Event-B context:**
   - **FR_i** → **FR_C**, where **FR_C** is the context of Event-B.
   - The context is defined with the **FR_i** as variables.
   - The context is represented as a set of rules in Event-B form.

2. **Translation of fuzzy requirements to Event-B machine:**
   - **FR_i** → **FR_M**, where **FR_M** is the machine of Event-B.
   - The machine is defined with the **FR_i** as rules.
   - The machine is represented as a set of transition rules in Event-B form.

**Translation rules:**

- Translation of fuzzy requirements to Event-B context:
  - **FR_i** → **FR_C**, where **FR_C** is a collection of variables and rules.
  - The context is defined with **FR_i** as variables.
  - The context is represented as a set of rules in Event-B form.

- Translation of fuzzy requirements to Event-B machine:
  - **FR_i** → **FR_M**, where **FR_M** is a collection of rules.
  - The machine is defined with **FR_i** as rules.
  - The machine is represented as a set of transition rules in Event-B form.

**Translation rules:**

- **FR_i** to **FR_C** transformation rules:
  - Translate fuzzy requirements to classical sets.
  - Represent fuzzy requirements in Event-B context.

- **FR_i** to **FR_M** transformation rules:
  - Translate fuzzy requirements to Event-B machine.
  - Represent fuzzy requirements as transition rules in Event-B machine.

**Translation rules:**

- **FR_i** to **FR_C** transformation rules:
  - Define variables and rules in Event-B context.
  - Represent fuzzy requirements in Event-B context.

- **FR_i** to **FR_M** transformation rules:
  - Define rules in Event-B machine.
  - Represent fuzzy requirements as transition rules in Event-B machine.

**Translation rules:**

- **FR_i** to **FR_C** transformation rules:
  - Define variables and rules in Event-B context.
  - Represent fuzzy requirements in Event-B context.

- **FR_i** to **FR_M** transformation rules:
  - Define rules in Event-B machine.
  - Represent fuzzy requirements as transition rules in Event-B machine.
• Rule 1. All hedges $\delta_i$, generators $Y_i$ and values $P_i$ in the collection of requirements are translated to three sets $\delta$, $Y$, 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 $\delta \times P$ (Corollary 5.1).

• Rule 4. Each requirement $FR_i$ is modeled by an event $ev_i$ in Event-B machine $FR_M$.

• Rule 5. Safety properties of the system are modeled as invariants $\mathcal{I}$ of the 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).

```
MACHINE FR_M
SEES FR_C
VARIABLES
  x_i
  m_i
INvariants
  inv1: x_i \in \mathcal{P}(\delta \times Y)
  inv2: m_i \in \mathcal{P}(\delta \times P)
  inv3: \mathcal{I}
EVENTS
Event FR_i \triangleq
  when
    grd1: x_i = \{\delta_i \mapsto Y_i\}
  then
    act1: m_i := \{\delta_i \mapsto P_i\}
    act2: x_i := \{\delta_j \mapsto Y_j\}
end
END
```

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

### 5.2.3 Modeling timed fuzzy systems

First, we define timed Fuzzy If-Then rules that has the form as follows: IF $x(t)$ is A THEN $y(t)$ is B. Following the approach presented in [4], if a fuzzy requirement $FR_i$ contains any time-dependent variable, then we will refine the appropriated event of the abstract machine (Subsection 5.2.2, Rule 5). 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 $\delta$ 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 $x_c \in \mathbb{R} \rightarrow \delta \times P$.

We introduce two rules for modeling continuous transitions the requirements as follows:
• Rule 6: If either variable $x_i$ or $m_i$ (in a fuzzy requirement $FR_i$) attaches to time-axis, 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 7: The time dependent variables (in Rule 1) are replaced by time functions and glue invariants in the refined machine.

5.3 A case study: Container Crane Control

5.3.1 Case study description

Container cranes are used to load and unload containers on a off ships in most harbors. 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.

5.3.2 Modeling Container Crane Control system

Modeling the discrete behavior:

Applying the translation rules presented in Subsection 5.2.2, 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 $h_{deg}(\text{very}) = 3 \land h_{deg}(\text{quite}) = 2 \land h_{deg}(\text{precise}) = 1$.

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 5: Each imprecise requirement $FR_i$ of the system is translated to an EVENT $evt_i$, $i = 1, 5$.

Modeling the continuous behavior:

We refine the translated model in the first part of this Subsection by having more closer to system. In fact, each movements 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 5.2.3 as follows:

• Apply Rule 6: Five events are refined in the refined machine Crane_M1, variable $t$ (time counter) is added.
• Apply Rule 7: Replace $dis$ 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 ($inv3$).

Chapter 6. Verifying imprecise system requirements

6.1 Introduction

Formal methods are used not only for specifying but also detecting and resolving conflicts of requirements. In Chapter 5, we propose a new method to model imprecise system requirements described by a set of Fuzzy If-Then rules in which functional requirements are translated into Event-B constructs. However, verification of significant properties of such systems are not discussed yet. 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. As far as we know, this paper reports the first concrete results of formal checking of such properties for imprecise system requirements. The contributions of this chapter are as follows: (1) providing a set of translation rules to formalize eventualities by Event-B language constructs, which makes use of the refinement modeling approach that Event-B supports, (2) demonstrating how both safety and eventuality properties of a set of the Fuzzy If-Then rules are verified with RODIN/Event-B.

6.2 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 $VAR$ ensures the decrement of the variant and $NAT$ makes sure that the variant is a natural number after the event execution.

6.3 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. Event-B does not support to specify liveness properties directly but we can follow the approach [6] to verify properties such as existence ($\square \Diamond P$), progress ($\square (P_1 \Rightarrow \Diamond P_2)$), persistence ($\Diamond \square P$), where $P$ is any first order logic formula, $\square$ and $\Diamond$ are standard operators of Linear Temporal Logic (LTL).
6.4 Modeling properties of fuzzy requirements

6.4.1 Modeling safety properties

Corollary 6.1. With the modeling proposed in transformation rules, the safety properties are preserved by all actions in imprecise requirements of the system.

6.4.2 Modeling eventuality properties

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 6. Fuzzy values of each element in \( P, Y \) and hedges \( \delta \) are translated to total functions \( \text{deg}_P \rightarrow \mathbb{N}, \text{deg}_Y : Y \rightarrow \mathbb{N}, \text{deg}_H : \delta \rightarrow \mathbb{N} \) respectively.
- Rule 7. Adds a variant mapping to linguistic variable that appears in eventuality property expression \( Q \).
- Rule 8. Refines each event representing for fuzzy if-then requirements by two events: a convergent and an ordinary one.
- Rule 9. 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 10. Deadlock free property is encoded as a theorem of the refined machine.

Definition 6.2 (Convergence). Fuzzy rules are convergent from a state \( Q(x) \) if each rule decreases value of variable \( x \). It is formally defined as:

\[
FR_i, Q(x) \vdash x' < x, \text{ where } x' \text{ is value after executing rule } FR_i.
\]

Definition 6.3 (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 6.4. With proposed modeling 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 \( \{FR\} \vdash \square\diamond \neg Q(x) \).

6.5 Verify the Container Crane Control system

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

6.5.1 Refinement: Modeling eventuality property

We perform the refinement strategy by following the method described in Section 6.4.2 to model the desired eventuality property. First, we apply Rule 6 to extend the abstract context \( \text{CraneCtrlC0} \) to define \( \text{CraneCtrlC1} \) by introducing three total functions for numerical values of fuzzy sets. We refine the abstract machine \( \text{CraneM0} \) to have \( \text{CraneM1} \) with five convergent events and five ordinary events (following Rule 8). The snippets below show event \text{evt4} only.
6.5.2 Checking properties

The system has a safety property which is formalized as an invariant clause
\( \text{inv}4 : \text{ran}(\text{dist}) = \{\text{close}\} \Rightarrow \neg \text{ran}(\text{power}) = \{\text{fast}\} \). Invariant preservation PO is generated for each event of the machine \( \text{Crane}_M0 \). Table 6.1 shows the invariant preservation PO for invariant \( \text{inv}4 \) of event \( \text{evt}4 \).

<table>
<thead>
<tr>
<th>\text{Table 6.1: INV PO of event \text{evt}4}</th>
<th>\text{evt4/inv4/INV}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{ran}(\text{dist}) = {\text{close}} \Rightarrow \neg \text{ran}(\text{power}) = {\text{fast}}</td>
<td>\text{evt4/inv4/INV}</td>
</tr>
<tr>
<td>\text{dis} = {\text{precise} \mapsto \text{close}}</td>
<td>\text{evt4/inv4/INV}</td>
</tr>
<tr>
<td>\neg \text{ran}({\text{precise} \mapsto \text{above}}) = {\text{close}} \Rightarrow \neg \text{ran}({\text{very} \mapsto \text{slow}}) = {\text{fast}}</td>
<td>\text{evt4/inv4/INV}</td>
</tr>
</tbody>
</table>

We have to prove that eventually the crane loader will reach above position of container, i.e. \( \text{Crane}_M1 \vdash \Box \Diamond (d = \text{deg\_DIS}(\text{above})) \). The deadlock-free property of this machine is encoded as the theorem \( \text{DELF} \) in \( \text{Crane}_M1 \). Its proof obligation is generated as \( \text{DELF}/\text{THM} \).

<table>
<thead>
<tr>
<th>\text{Table 6.2: Deadlock free PO of machine \text{Crane}_M1}</th>
<th>\text{DELF/THM}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{\text{dist} = {\text{precise} \mapsto \text{close}}}</td>
<td>\text{DELF/THM}</td>
</tr>
<tr>
<td>\neg \text{ran}({\text{precise} \mapsto \text{above}}) = {\text{close}} \Rightarrow \neg \text{ran}({\text{very} \mapsto \text{slow}}) = {\text{fast}}</td>
<td>\text{DELF/THM}</td>
</tr>
</tbody>
</table>

In order to check the convergent property, proof obligations are generated for each convergent events of machine \( \text{Crane}_M1 (\text{evt}_i/\text{NAT} \text{ and } \text{evt}_i/\text{VAR}) \). Table 6.3 is the proof obligation that shows event \( \text{evt}4 \) of machine \( \text{Crane}_M1 \) decrease variant \( d \).
Chapter 7. Conclusions

7.1 Conclusions
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. 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.

7.2 Limitations
- 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.

### 7.3 Future work

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. Therefore, an enhancement that makes it possible to prove 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.
Bibliography


LIST OF PUBLICATIONS


