VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Lê Hồng Anh

METHODS FOR MODELING AND VERIFYING EVENT-DRIVEN SYSTEMS

Major: Software Egnineering
Code: 62.48.01.03

SUMMARY OF DOCTORAL THESIS IN INFOMRATION TECHNOLOGY

Hà Nội – 2015#

#

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

Supervisors: Assoc. Prof. Dr. Trương Ninh Thuận

Assoc. Prof. Dr. Phạm Bảo Sơn

Reviewer: Assoc. Prof. Dr. Nguyễn Đình Hóa.....

.....

Reviewer: Assoc. Prof. Dr. Huỳnh Quyết Thắng. .....

.....

Reviewer: Dr. Nguyễn Trường Thắng.....

.....

The thesis defence takes place at …

This thesis can be found at:

- National Library of Vienam
- Library and Information Center - VNU

#

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. 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. 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 new and effective approaches 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 directs at providing tools which support for automatic translation to make less effort in modeling process. 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

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 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.
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 structured in Figure 1.1. 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 present new modeling methods for modeling and verifying event-driven systems described by imprecise requirements. Chapter 6 summarizes the thesis and present future work.

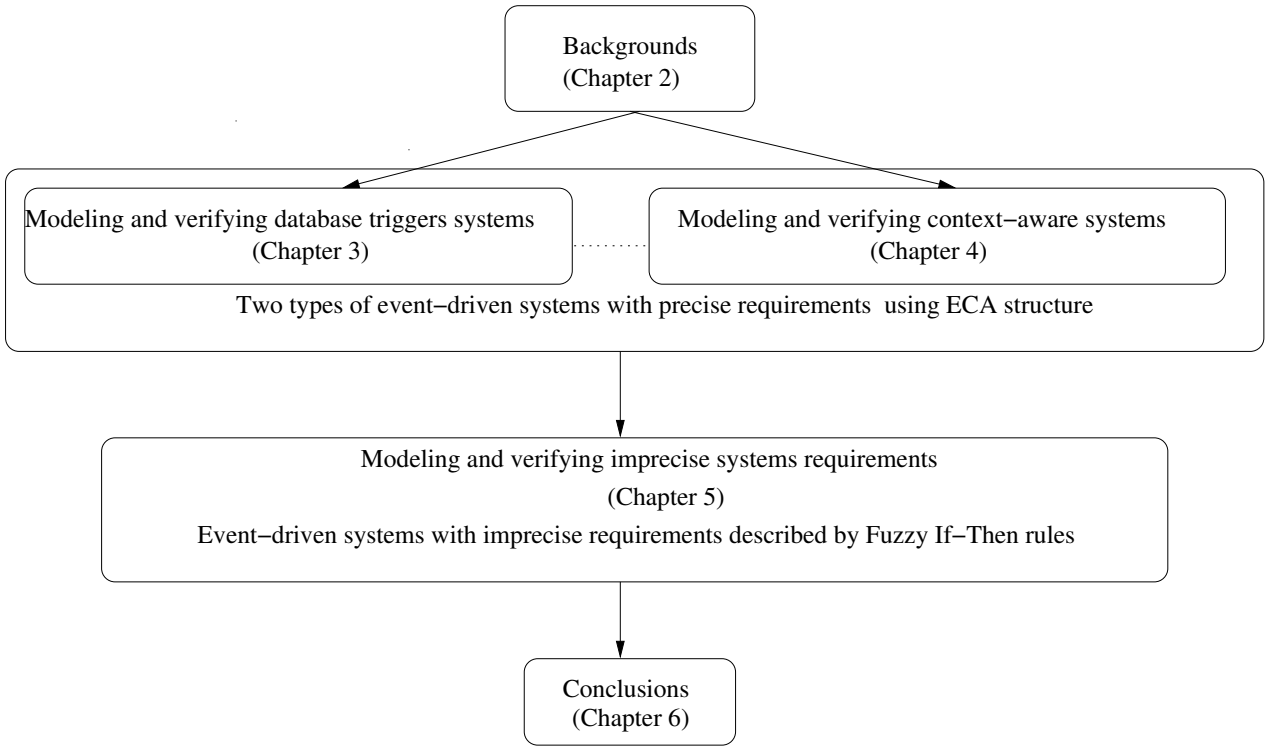

FIGURE 1.1: Thesis structure

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 such as $\wedge$, $\vee$, and $\neg$; and the "temporal" operators such as $\Box$ ("always"),$\Diamond$ ("eventually"), $\circ$ ("next"), and $\mathcal{U}$ ("until").

2.2 Classical set theory

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. There are some basic definitions of set theory such as power set, relations, functions, etc.

2.3 Fuzzy sets and Fuzzy If-Then rules

In order to deal with systems which are too complex or too ill-defined to admit of precise descriptions, Zadeh 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 represented as a pair as follows: $F = \{(x, \mu_F(x))\}$ where $x \in X$ and $\mu_F(x) : X \rightarrow [0, 1]$ is termed as the grade of membership of x in F. 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. 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 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. 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, B.

2.4.1 VDM

VDM stands for "The Vienna Development Method" which is a model-based method giving descriptions of software systems and other systems as models. 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

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.

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. 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 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. Events $E(v)$ present transitions between states. Each event has the form evt = 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: (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. 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 (DLKF).

2.6 Rodin tool

This thesis uses the RODIN toolkit version 2.8 which is an Eclipse environment for modeling and proving in Event-B. The Rodin tool provides a rich of perspective windows to users to develop Event-B models easily. The proof obligations of the model are generated automatically. Rodin provides both automatic proving and interactive proving mechanisms.

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. In this thesis, we consider two applications of active objects and rule-based production systems: active databases and context-aware systems.

2.7.1 Database systems and database triggers

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, 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 reacts to the context changes.

Chapter 3. Modeling and verifying database trigger 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. 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 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 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	$t = \langle r_1, .., r_m \rangle$	$T = TYPE_1 \times TYPE_2 \times TYPE_n$
Rule 3	$r_i = \langle f_{i1}, .., f_{in} \rangle$	$t \in \mathbb{P}(T)$
Rule 4	Primary key constraint	$f : TYPE_1 \nrightarrow T$
Rule 5	Constraint C	Invariant I
Rule 6	Trigger E	Event Evt

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

TABLE 3.2: Formalizing a trigger by an Event-B Event

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

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 the state of the system does not change after a sequence of events. 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

TABLE 3.3: Encoding trigger actions

INSERT INTO T VALUES (value1,..,valuen)	ANY r WHEN $(r \in T \wedge e \wedge c)$ THEN $T := T \cup r$ end
DELETE FROM T WHERE $\langle column1 = some\_value \rangle$	ANY v WHEN $(v \in TYPE_1 \wedge e \wedge c)$ THEN $t := t - f(v)$ end
UPDATE T SET column1=value, column2=value2 WHERE $\langle column1 = some\_value \rangle$	ANY $v1, v2$ WHEN $v1 \in TYPE_1 \wedge v2 \in TYPE_2 \wedge e \wedge c$ THEN $t := \{1 \mapsto value1, 2 \mapsto value2\} \oplus t$ end

freedom rule is stated as $I(v), P(c) \vdash G_1(v) \vee ... \vee G_n(v)$, 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.

- Constraint preservation: With the proposed translation method, a trigger does not break these rules if $I(v), G(w, v), S(w, v, v') \vdash I(v')$. This is also the INV proof obligation of Event-B events.

3.3 A case study: Human resources management application

3.3.1 Scenario description

A database system of a human resource application has two tables EMPLOYEES (including id and level columns) and BONUS (including id and amount columns). The database system has a constraint: *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 scenario

The Event-B specification of the example is partly shown in Figure 3.1, Figure 3.2, 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 $= \mathbb{N} \times \mathbb{N}$
  axm3 : TBL_BONUS $= \mathbb{N} \times \mathbb{N}$
  axm4 : partition(TABLE_NAMES, {employees}, {bonus})
END

FIGURE 3.1: A part of Event-B Context

3.3.3 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 $trigger1$ is illustrated in Table 3.4. Two events

MACHINE *TRIGGER_M*
SEES *TRIGGER_C*
VARIABLES
bonus
empl
f_bonus
f_empl
type
INVARIANTS
inv1 : $bonus \in \mathbb{P}(TBL\_BONUS)$
inv2 : $empl \in \mathbb{P}(TBL\_EMPL)$
inv3 : $type \in TYPES$
inv4 : $f\_bonus \in \mathbb{N} \pfun \mathbb{N}$
inv5 : $f\_empl \in \mathbb{N} \pfun \mathbb{N}$
SYS_CTR : $\forall eid.eid \in dom(empl) \land pk\_empl(eid) > 5 \Rightarrow pk\_bonus(eid) > 10$
INF_LOOP : $(type = update \land table = BONUS) \lor (type = update \land table = EMPL)$
END

FIGURE 3.2: A part of Event-B machine

Event *trigger1* $\widehat{=}$
any
eid
when
grd1 : $type = update$
grd2 : $table = EMPL$
grd3 : $eid \in dom(empl)$
then
act1 : $type := update$
act3 : $table := BONUS$
act5 : $bonus := \{eid \mapsto (pk\_bonus(eid) + 10)\} \oplus bonus$
act5 : $pk\_bonus(eid) := pk\_bonus(eid) + 10$
end
Event *trigger2* $\widehat{=}$
any
eid
when
grd1 : $type = update$
grd2 : $table = BONUS$
grd3 : $pk\_bonus(eid) \geq 10$
then
act1 : $type := update$
act2 : $table := EMPL$
act3 : $empl := \{eid \mapsto (pk\_empl(eid) + 1)\} \oplus empl$
end

FIGURE 3.3: Encoding trigger

*Trigger*1 and *Trigger*2 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's guards is added to the target machine. If we show that this invariant is preserved by machine *DB_M*0, then two triggers execution

TABLE 3.4: Proof obligation of constraint preservation

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

leads to the infinite loop. The proof clause of the event $trigger1$ is presented in Table 3.5.

TABLE 3.5: Proof obligation of innifte loop

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

3.4 Support tool: Trigger2B

Following the method presented in Section 3.2, we implement a supporting tool called Trigger2B that generates multiples XML-based format as the output. These files are then able to be used in the support tool Rodin.

3.4.1 Architecture

The architecture of this tool consisting five modules is illustrated in Figure 3.4.

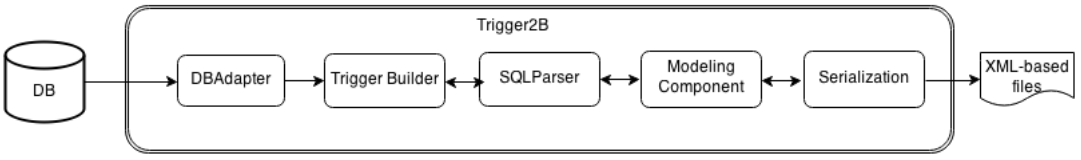

FIGURE 3.4: Architecture of Trigger2B tool

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, 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 tasks 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.

3.4.2 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 put of SQLParser component which currently uses ANLTR framework to parse sql statements. A parsed tree of general triggers is partially illustrated in Figure 3.5.

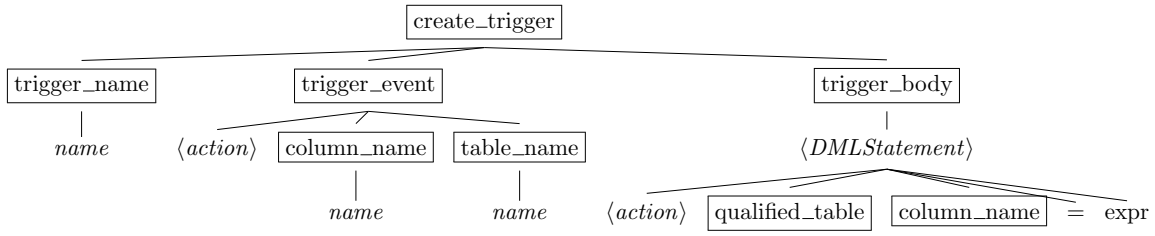

FIGURE 3.5: A partial parsed tree syntax of a general trigger

We propose an algorithm following our proposed translation rules to transform the parsed tree to an Event-B model. The algorithm is illustrated as follows:

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

```
1   begin
2       node = root(t)
3       while (isVisited(node))
4           if node.type = create_trigger then
5               e=createNewEvent(M)
6               if node.type = trigger_name then
7                   e.name = node.name
8               elseif node.type = trigger_event
9                   for child in nodes.childs
10                      if node.type = action then
11                          addGuard(e,type=node.value)
12                      if node.type = tabletable_name then
13                          addGuard(e,table=node.child.value)
14              elseif node.type = trigger_body
15                          addAction(e,getExp(node.childs))
16          end
17          visit_next(node)
18      end
```

Chapter 4. Modeling and verifying context-aware systems

4.1 Introduction

Context awareness of an application relates to adaptation, responsiveness, sensitiveness of the application to changes of the context. 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

TABLE 4.1: Transformation between context-aware systems and Event-B notations

	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

4.2.2 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 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.

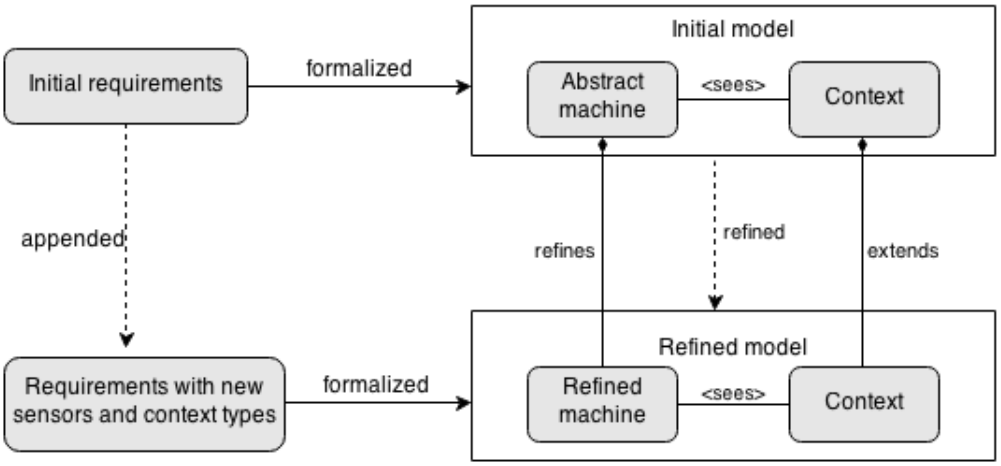

FIGURE 4.1: Incremental modeling using refinement

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.

- 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 uses 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 increases 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).

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

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 4.2: 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:

- 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*"

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

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
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

FIGURE 4.3: Refined Event-B model for ACC system

TABLE 4.2: Proof of context constraint preservation

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

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. 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, 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 or Z notation.

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 by Event-B language constructs, 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.

5.2 Modeling fuzzy requirements

5.2.1 Representation of fuzzy terms in classical sets

Corollary 5.1. *A collection of well-defined fuzzy requirements "If x is δY then m is γP" can be specified by classical sets.*

5.2.2 Modeling discrete states

We propose below partial transformation rules to map fuzzy requirements to Event-B's elements.

- Rule 1. All hedges δ_i, 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 5.1).

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

Figure 5.1 illustrates the Event-B specification after applying the translation rules.

CONTEXT *FR_C*
SETS
γ
δ
P
Y
END

MACHINE *FR_M*
SEES *FR_C*
VARIABLES
x_i
m_i
INVARIANTS
inv1 : $x_i \in \mathbb{P}(\delta \times Y)$
inv2 : $m_i \in \mathbb{P}(\gamma \times P)$
EVENTS
Event $FR_i \mathrel{\widehat{=}}$
when
grd1 : $x_i = \{\delta_i \mapsto Y_i\}$
then
act1 : $m_i := \{\gamma_i \mapsto P_i\}$
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 continuous behavior

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 by J.Abrial, if a fuzzy requirement FR_i contains any time-dependent variable, then we will refine the appropriated event of the abstract machine. 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) 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 6: The time dependent variables (in Rule 2) are replaced by time functions and glue invariants in the refined machine.

5.3 Verifying safety and eventuality properties

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

5.3.2 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. Event-B does not support to specify liveness properties directly but we can apply some recent research results 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).

5.3.3 Modeling 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.

Corollary 5.2. *With the modeling proposed in transformation rules, the safety properties are preserved by all actions in imprecise requirements of the system.*

5.3.4 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 7. Fuzzy values of each element in P, Y and hedges δ are translated to total functions $deg_P \rightarrow \mathbb{N}$, $deg_y\, Y : Y \rightarrow \mathbb{N}$, $deg_H : \delta \rightarrow \mathbb{N}$ 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.

Definition 5.3 (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 5.4 (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 5.5. *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* $\{FR\} \vdash \Box\Diamond\neg Q(x)$.

5.4 A case study: Container Crane Control

5.4.1 Case study description

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 start position (property $\mathcal{Q}$). Then we have to check if $\{FR\} \vdash \mathcal{I}$ and $\{FR\} \vdash \Box\Diamond\neg\mathcal{Q}$.

5.4.2 Modeling Container Crane Control system

5.4.2.1 Modeling the discrete behavior:

- 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 $h\_deg(very) = 3 \wedge h\_deg(quite) = 2 \wedge h\_deg(precise) = 1$.
- 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, $i = \overline{1,5}$.

5.4.2.2 First refinement: Modeling the continuous behavior:

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 proposed rules to model the continuous behavior as follows:

- Apply *Rule 5*: Five events are refined in the refined machine $Crane\_M1$, variable t (time counter) is added.

- Apply *Rule 6*: 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 ($inv3$).

5.4.2.3 Second refinement: Modeling eventuality property

We perform the refinement strategy by following the method described in Section 5.3.4 to model the desired eventuality property. First, we apply Rule 7 to extend the abstract context *Crane_C0* to define Crane_C1 by introducing three total functions for numerical values of fuzzy sets. We refine the abstract machine Crane_M1 to have Crane_M2 with five convergent events and five ordinary events (following Rule 9). The snippets below show event $evt4$ only.

CONTEXT Cranel_C1
EXTENDS Crane_C0
CONSTANTS

deg_HED, deg_POWER, d_DIS

AXIOMS

axm4 : deg_HED : HEDGES $\rightarrow \mathbb{N}$

axm5 : deg_HED(very) = 3 $\wedge$ deg_HED(quite) = 2
$\wedge$ deg_HED(precise) = 1

END

Event $evt4\_CE \mathrel{\widehat{=}}$
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 $evt4\_OE \mathrel{\widehat{=}}$
Status ordinary
extends evt_4
when
grd1 : distance = {precise $\mapsto$ close}
grd2 : d = deg_DIS(close)
grd3 : d = deg_DIS(above)
then
act1 : power := {very $\mapsto$ slow}
act2 : distance := {precise $\mapsto$ above}
act2 : d := deg_DIS(above)
end

5.4.3 Checking properties

The system has a safety property which is formalized as an invariant clause $inv4 : ran(dist) = \{close\} \Rightarrow \neg ran(power) = \{fast\}$. 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 $evt4$

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

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 $Crane\_M1$

$d = deg\_DIS(above)$ $\Rightarrow$ $d = deg\_DIS(start) \vee d = deg\_DIS(far)$ $d = deg\_DIS(medium) \vee d = deg\_DIS(close)$ $d = deg\_DIS(above)$	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.

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

Chapter 6. Conclusions

6.1 Achievements

The research results of the thesis achieved the defined objectives. 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. A tool called Trigger2B is also developed to support for automatically translating a database system to an Event-B model.

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.

6.2 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.

6.3 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 and 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. Vol2, e4, 2014. ISSN 2409-0026.

6. Hong Anh Le, Ninh Thuan Truong and Shin Nakajima: Verifying Eventuality Properties of Imprecise System Requirements using Event-B. In Proc. The 30th ACM/SIGAPP Symposium On Applied Computing - Software Engineering Track, Salamanca, Spain. Volume 2, pp. 1651-1654, ACM Press.