


Formal Analysis of Database Trigger Systems Using Event-B

Anh Hong Le, Hanoi University of Mining and Geology, Hanoi, Vietnam

 <https://orcid.org/0000-0002-0483-3195>

To Van Khanh, VNU University of Engineering and Technology, Hanoi, Vietnam

Truong Ninh Thuan, VNU University of Engineering and Technology, Hanoi, Vietnam

ABSTRACT

Most modern relational database systems use triggers to implement automatic tasks in response to specific events happening inside or outside a system. A database trigger is a human readable block code without any formal semantics. Frequently, people can check if a trigger is designed correctly after it is executed or by manual checking. In this article, the authors introduce a new method to model and verify database trigger systems using Event-B formal method at design phase. First, the authors make use of similar mechanism between triggers and Event-B events to propose a set of rules translating a database trigger system into Event-B constructs. Then, the authors show how to verify data constraint preservation properties and detect infinite loops of trigger execution with RODIN/Event-B. The authors also illustrate the proposed method with a case study. Finally, a tool named Trigger2B which partly supports the automatic modeling process is presented.

KEYWORDS

Event-B, Modeling, Trigger, Verification

1. INTRODUCTION

Traditional database management systems (DBMS) are passive as they execute commands when applications or users perform appropriate queries. The research community has rapidly realized the requirement for database systems to react to data changes. Most modern relational databases include these features as triggers (or active rules) that monitor and react to specific events happening inside and outside of a system. They also use triggers to implement automatic tasks when a predefined event occurs.

DBMS usually have two types of triggers: data manipulation language (DML) and system triggers. The former is fired whenever the DML statements such as *deleting*, *updating*, and *insert* statements are executed, the latter is performed in case that system events or data definition language (DDL) ones occur. A trigger has the form of an Event-Condition-Action (ECA) rule informally written as “if a set of *events* occur and a set of *conditions* hold, then perform *actions*”. It is made of a block of code and has syntax, for example, an Oracle trigger is similar to a stored procedure containing blocks of PL/

DOI: 10.4018/IJSI.20211001.oa1

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

SQL code. Trigger codes are understanding semantic and do not have any formal semantic. Trigger execution may lead to an infinite loop when triggers call each other or it may violate database system constraints. We can only check these properties after executing triggers. In fact, it is valuable if we can show that trigger execution is correct at design phase because it reduces development cost for database design. Thus, a formal framework for modeling and verifying database triggers is desirable.

Many researchers have been working on analyzing triggers (or active rules). The research results of Lee and Ling (1998, 1999), proposed in the early 1990s, transformed ECA rules to some types of graphs and applied various static analysis techniques to check properties such as *redundancy*, *inconsistency*, *incompleteness*, and *termination*. Baralis (1999) proposed a technique, based on relational algebra, to check if active rules are terminated or confluent. Other results Choi et al (2006b); Chavarria-Báez and Li (2007) addressed both *termination* and *confluence* properties using model checking techniques. One important property that has not received much attention is data constraint property of a system. A terminated trigger still can cause critical problems if it violates data constraints. Furthermore, a method or an approach with supporting tools, which is feasible to apply in database development to check both data constraints and infinite loops, is also desirable.

Our previous work Le and Truong (2013) initially proposed to use Event-B to formalize and verify a database triggers system at design phase. The main idea of the method comes from the similar structure and working mechanism of Event-B events and database triggers. We presented a set of translation rules to translate a database system including triggers to an Event-B model. In this paper, we make the translation rules more precise when encoding the body of trigger. With the proposed modeling method, we can formally check if a system preserves the data constraints and find infinite loops by proving the proof obligations of the translated Event-B model. The advantage of our method is that a database system including triggers and constraints can be modeled naturally by Event-B constructs such as *invariants* and *events*. As far as we know, this paper reports the first concrete result of analyzing a database system with DML triggers using formal method. We also developed a tool called Trigger2B which partly supports automatic modeling process with the RODIN. We introduce an algorithm to construct an Event-B model from a syntax tree of triggers written in SQL. In the supporting tool RODIN, almost proofs are discharged automatically, hence it reduces complexity in comparison with manual proving. The tool makes the proposed method feasible in database development process.

The remainder of this paper is organized as follows. Section 2 provides basic knowledge of database triggers and Event-B. In Section 3, we present our proposed method to model and verify database systems including triggers. Section 4 introduces a scenario of a human resource management application to demonstrate the method in detail. We present the tool Trigger2B, which supports for partly automatic translation in Section 5 and we summarize related work in section 6. We conclude the paper and present the future work in Section 7.

2. PRELIMINARIES

In this section, we first briefly introduce database triggers and their SQL syntax. Then we give an overview of Event-B formal method.

2.1. Database Triggers

A relational database system, 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 or outside of database.

A database trigger is a block code that is automatically fired in response to a defined event in a database. The defined 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, and to implement complex business rules. The structure of a trigger follows ECA structure, hence it takes the following form:

It means that whenever event e occurs and the *condition* is met then the database system performs *actions*. Database triggers can be mainly classified by two kinds: Data Manipulation Language (DML) and Data Definition Language (DDL) triggers. The former is executed when data is manipulated, while in some database systems, the latter is fired in response to DDL events such as creating table or events such as login, commit, and roll-back, etc. Users of some relational database systems such as Oracle, MySQL, and SyBase are familiar with triggers which are represented in SQL:1999 format Eisenberg and Melton (1999) (the former is SQL-3 standard). The definition of SQL:1999 trigger has syntax as follows:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
{BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON <table_name>
[REFERENCING [NEW AS<new_row_name>] [OLD AS<old_row_name>]]
[FOR EACH ROW [WHEN (<trigger_condition>)]]
<trigger_body>
```

2.2. Event-B

Event-B Abrial (2010) 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 in 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 relevant properties and hypotheses are defined. A CONTEXT consists of carrier sets, constants, and 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 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. Invariants $I(v)$ yield laws that state variables v must be always 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 “ $evt = \text{any } x \text{ where } G(x, v) \text{ then } A(x, v, v') \text{ end}$ ”, where x represents 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 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 systems gradually by adding more details to construct a 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 models. In superposition refinement, abstract variables are retained in a concrete machine, with possibly some additional variables. In vertical refinement such as data refinement, the abstract variable v is replaced by concrete ones w . Subsequently, connections between them are represented by a 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 must be proven. Two of the proof obligations mainly focused are invariant preservation (INV), deadlock-freeness (DLF). INV PO means that proving invariants hold after execution of events. The proof obligation is as follows: $I(v), G(w, v), S(w, v, v') \vdash I(v')$ where $I(v)$ denotes the invariant, $G(w, v)$ is the guard clause of the events, and $S(w, v, v')$ is assignment. Deadlock-freeness for a machine ensures that there are always some enabled events during its execution. Assume that

a machine contains a set of n events $e_i \left(i \in \overline{1 \cdot n} \right)$ of the following form: $evt = \text{any } x \text{ where } G(x, v)$ then $A(x, v, v')$ end. The proof obligation rule for deadlock-freeness is as follows:

$$I(v) \vdash \bigvee_{i=1}^n \left(\exists x_i \cdot G(x_i, v) \right)$$

3. MODELING AND VERIFYING DATABASE TRIGGER SYSTEM

As stated above, it is important to check that if a database trigger system is designed correctly at design phase. In this section, we introduce a new method to model and verify a database system including triggers using Event-B. The method allows detecting infinite loops and gives a warranty of data constraint preservation.

3.1. Modeling Database Systems

A database system is normally composed of 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 data constraints. Before modeling a database system by Event-B, we introduce some database definitions in set theory which are the basis for modeling process.

Definition 1: (Database system) - A database system is modeled by a 3-tuple $db = \langle T, C, G \rangle$, where T is a set of table, C states system constraints, and G indicates a collection of triggers.

Definition 2: (Table) - For each $t \in T$, denoted by a tuple $t = \langle r_1, \dots, r_m \rangle$, where m is the total number of rows in the table t , and r_i is a set indicating the i -th row of the table $\left(i \in \overline{1 \cdot m} \right)$. A row is stated by a tuple $r_i = \langle f_{i1}, \dots, f_{in} \rangle$, where n is the total number of columns, f_{ij} represents data of column j at row $i \left(j \in \overline{1 \cdot n} \right)$.

Definition 3: $\langle \text{Trigger} \rangle$ - Each trigger $g \in G$ is denoted by a 3-tuple $g = \langle e, c, a \rangle$, where e is a type of the trigger's event, c is a condition of the trigger, and a is an action of the trigger.

Based upon these definitions, we present a set of translation rules to translate a database model to an Event-B model illustrated in Table 1. These rules are described in detail as shown in Table 1.

Rule 1: A database system is formalized by a pair of Event-B machine and context: DB_M, DB_C

Rule 2: A table is presented by a Cartesian product of N sets $T = TYPE_1 \times TYPE_2 \times \dots \times TYPE_n$ where $TYPE_i$ denotes data type of column i . To manipulate table T , we add a variable $t \in \rho(T)$ to the machine DB_M .

Rule 3: Each table T has a primary key constraint. We encode this kind of constraints as a bijective function:

$$f \in TYPE_1 \times TYPE_2 \times \dots \times TYPE_i \rightarrow \left(TYPE_{i+1} \times TYPE_{i+2} \times \dots \times TYPE_n \right)$$

we assume that the first i columns of the table form the primary key.

Rule 4: A data constraint C is formalized by an invariant I .

Table 1. Translation rules between database and Event-B

	Database Definitions	Event-B Concepts
Rule 1	$db = T, C, G$	DB_M, DB_C
Rule 2	$r_i = f_{i1}, \dots, f_{im}$ $t = r_1, \dots, r_m$	$T = TYPE_1 \times TYPE_2 \times \dots \times TYPE_n$
Rule 3	Primary key constraint	$f \in TYPE_1 \times \dots \times TYPE_i \rightarrow TYPE_{i+1} \times \dots \times TYPE_n$
Rule 4	Constraint C	Invariant i
Rule 5	Trigger E	Event evt

Rule 5: A trigger E is translated to an event Evt .

Example: Let assume that a database system M consists of two tables $T1, T2$ (both of them have two columns), two triggers G_1, G_2 and one data constraints C . The Event-B specification of the system is partially described in Figure 1.

3.2. Formalizing Triggers

In this Section, we show in detail how to formalize database triggers. Recall that, a trigger is denoted by 3-tuple $g = \langle e, c, a \rangle$, where e is a type of the trigger, c is a condition in which the trigger happens,

Figure 1. Partial Event-B specification of database system M

```

CONTEXT DB_C
CONSTANTS
    T1
    T2
AXIOMS
    axm1 : T1 = {TYPE1 × TYPE2}
    axm2 : T2 = {TYPE3 × TYPE4}
END

MACHINE DB_M
SEES DB_C
VARIABLES
    t1
    t2
    f1f2
INVARIANTS
    inv1 : t1 ∈ P(T1)
    inv2 : t2 ∈ P(T2)
    inv3 : f1 ∈ TYPE1 ⇒ TYPE2
    inv3 : f2 ∈ TYPE3 ⇒ TYPE4
    inv4 : I
EVENTS
    Event G1 =
        ...
    Event G2 =
        ...
END

```

and a is trigger's actions. As illustrated in Table 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. Actions of a trigger are translated to the body part of an Event-B event.

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

Table 2. Formalizing a trigger by an Event-B Event

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

A trigger action is a block code and its syntax depends on database management systems. This block code also contains SQL statements. In order to show how our method works, we simplify the case by considering that the Action part of a trigger contains a sequence of DML statements without branch or loop statements. Hence, the action of a trigger consists of a sequence of *Insert*, *Update* or *Delete* statement. The rules translating the *Action* part of a trigger is described as follows:

- **Insert:** This statement has the form “Insert into T values (val_1, \dots, val_n)”, where val_1, \dots, val_n are column values of the new record of the table T . We encode this new row as a parameter $r \in T$ of the event. More specifically, the translated event has the form $Evt = \mathbf{Any} \ r \ \mathbf{Where} \ r \in T \wedge e \wedge c \ \mathbf{Then} \ t := t \cup r$;
- **Delete:** This statement is generally written in the form: “Delete from T where $column_1 = some_value$ ”. It will delete the record that has the first column's value equal to *some_value*. We add a parameter for the event representing the value *some_value*. The event is specified in detail as follows $Evt = \mathbf{Any} \ v \ \mathbf{when} \ v \in TYPE_1 \wedge e \ \mathbf{Then} \ t := t - f(v)$;
- **Update:** The general syntax of this statement is “Update T set $column_1 = value_1$, $column_2 = value_2$ where $column_1 = some_value$ ”. This statement will update a record where the value of the first column is equal to *some_value*. Similar to the case of delete statement, we encode the input values as parameters of the event. The description of the translated event is as follows: $Evt = \mathbf{Any} \ v_1, v_2 \ \mathbf{when} \ v_1 \in TYPE_1 \wedge v_2 \in TYPE_2 \ \mathbf{Then} \ t := \{1 \mapsto v_1, 2 \mapsto v_2\} \oplus t$.

The translation is summarized in Table 3.

3.3. Verifying System Properties

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

- **Infinite Loop:** Since a trigger can fire other triggers, hence it probably leads to infinite loop. This situation occurs when after a sequence of events, the state of the system does not change. There are two ways to check this property of the system. The first one uses deadlock-freeness (DLKF) proof obligation of Event-B which states that the disjunction of the event guards always holds under the properties of the constant and the invariant.

Table 3. Encoding trigger actions

INSERT INTO T VALUES (val ₁ ,...,val _n)	ANY r WHEN $(r \in T \wedge e \wedge c)$ THEN $T := T \cup r$ END
DELETE FROM T WHERE $column_1 = some_value$	ANY v WHEN $(v \in TYPE_1 \wedge e \wedge c)$ THEN $t := t - f(v)$ END
UPDATE T SET $column_1 = value_1, column_2 = value_2$ WHERE $column_1 = some_value$	ANY v_1, v_2 WHEN $v_1 \in TYPE_1 \wedge v_2 \in TYPE_2 \wedge e \wedge c$ THEN $t := \{1 \rightarrow value_1, 2 \rightarrow value_2\} \oplus t$ END

The deadlock freedom rule is stated as $I, P(c) \vdash G_1(v) \vee \dots \vee G_n(v)$, where v is variable, $I(v)$ denotes invariant, $G_i(v)$ presents guard of the event. At the moment, the DLKF proof obligation is not generated automatically by the RODIN tool yet. However, we can generate it manually by adding a theorem saying the disjunction of guards. In some cases, DLKF theorem cannot be deduced from a set of invariants $I(v)$ 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 is always true before and after event execution including *INITIALISATION* event;

- **Constraint Preservation:** Data constraints are rules that the system should always conform. It means that before and after an execution of triggers, these properties hold. In Section 3, data constraints are expressed by invariants (I) and triggers are formalized by events $E(v)$. Hence, we prove that a trigger does not break these rules by proving $I(v) \wedge G(w, v) \wedge A(w, v, v') \vdash I(v')$. This is exactly as same as INV proof obligations of Event-B machine. Consequently, if these proof obligations are discharged then data constraint properties are satisfied.

4. A CASE STUDY

In this Section, we illustrate our proposed method on an extracted scenario of a human resource management application. We first describe the scenario where we use triggers, after that, we translate the scenario into an Event-B model and verify its properties.

4.1. Scenario Description

Let's assume that we have a database system of the human resource management application which includes two tables EMPLOYEES and BONUS structured in Table 4.

As a business requirement, the application needs to assure a condition as follows: *The bonus of an employee with a level greater than 5 is at least 5*. Hence, the database should always satisfy this constraint. The database system uses two triggers to do the following automatic tasks:

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

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

Table 4. Two tables employees and bonus

Employees		Bonus	
E Id	Level	E Id	Amount
0911	2	0911	2
0912	2	0912	2
0913	4	0913	4

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

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

4.2. Modeling the Scenario

We apply the method presented in Section 3 for modeling the system as follows:

- **Applying Rule 1:** The database system is formalized by a context *Trigger_C* and a machine *Trigger_M*, where *Trigger_C* contains a set *TYPE* representing all kinds of trigger;
- **Applying Rule 2:** Two constant sets *TBL_EMPL* and *TBL_BONUS* representing on two table *employees* and *bonus* respectively. Each table has two columns, hence each set is a Cartesian product of two sets \mathbb{N} (Figure 2). Variables *bonus* and *empl* are added into the machine *Trigger_M*.
- **Applying Rule 3:** Two bijective functions *f_bonus* and *f_empl* represent primary key relationship of table *bonus* and *employees* respectively.
- **Applying Rule 4:** The system constraint is formalized by invariant *SYS_CTR*.
- **Applying Rule 5:** Two triggers are translated to two events *trigger1* and *trigger2*.

Figure 2. A part of Event-B context

```
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
```


The specification of the machine is illustrated partly in Figure 3.

Next, we formalize two triggers of the system following the method presented in Section 3.2. In this case, two triggers are *update* triggers and are formally specified in Figure 4.

Figure 3. A part of Event-B machine

```

MACHINE TRIGGER_M
SEES TRIGGER_C
VARIABLES
    Bonus
    Empl
    f_bonus
    f_empl
    type
INVARIANTS
    inv1 : bonus ∈ P (TBL_BONUS)
    inv2 : empl ∈ P (TB_EMPL)
    inv3 : type ∈ TYPES
    inv4 : f_bonus ∈ N ⇒ N
    inv5 : f_empl ∈ N ⇒ N
    SYS CTR : ∀ eid · eid ∈ dom(empl) ∧ f_empl(eid) > 5
    ⇒ f_bonus(eid) > 10
    INF LOOP : (type = update ∧ table = BONUS) ∨ (type =
    update ∧ table = EMPL)
END

```

Figure 4. Encoding trigger

```

EVENT trigger1 =
ANY
    eid
WHEN
    grd1 : type = update
    grd2 : table = EMPL
    grd3 : eid ∈ dom(empl)
Then
    act1 : type := update
    act2 : table := BONUS
    act3 : bonus := {eid ↦ (f_bonus(eid) + 10)} ⊕ bonus
    act4 : pk_bonus(eid) := pk_bonus(eid) + 10
End

EVENT trigger2 =
ANY
    eid
WHEN
    grd1 : type = update
    grd2 : table = BONUS
THEN
    act1 : type := update
    act2 : table := EMPL
    act3 : empl := {eid ↦ (f_empl(eid) + 1)} ⊕ empl
End

```

4.3. Checking Properties

After having the translated Event-B model from the database system descriptions, it is possible to check if two triggers execution violates the data constraint property or leads to an infinite loop as follows:

- **Constraint Preservation:** Since the constraint property of the system is modeled by the invariant *SYS_CTR*:

$$\forall eid \cdot eid \in \text{dom}(\text{empl}) \wedge f_empl(eid) > 5 \Rightarrow f_bonus(eid) > 10$$

We need to prove that the invariant is maintained before and after events execution. The proof obligation of trigger1 is illustrated in Table 5. Two proof obligations (“trigger1/SYS_CTR/INV” and “trigger2/SYS_CTR/INV”) are automatically generated for two events (*trigger1* and *trigger2*) with the RODIN.

Table 5. INV PO of event trigger1 to verify data constraint property

$\forall \text{id}.\text{id} \in \text{dom}(\text{empl_rec}) \wedge f_empl(\text{id}) > 5 \Rightarrow f_bonus(\text{id}) > 10$ $\text{emplid} \in \text{dom}(\text{empl_rec})$ $\text{type} = \text{update}$ $\text{table} = \text{EMPL}$ \vdash $\forall \text{id}.\text{id} \in \text{dom}(\text{empl_rec}) \wedge f_empl(\text{id}) > 5$ $\Rightarrow (f_bonus \oplus \{\text{emplid} \mapsto f_bonus(\text{emplid}) + 10\})(\text{id}) > 10$	trigger1/ SYS CTR/ INV
--	------------------------------

These proof obligations are also automatically discharged. Consequently, the data constraint preserved by two triggers are translated to two events *trigger1* and *trigger2*.

- **Infinite Loop:** As we proposed in Section 3.3, an invariant *INF_LOOP* which is the disjunction of the event’ guards added to the target machine. If we show that this invariant is preserved by machine *DB_M0*, then execution of two triggers leads to an infinite loop. The proof clause of the event *trigger1* is presented in Table 6.

Two *INV* proof obligations are also generated and discharged automatically in the RODIN, i.e. the invariant clause is proved to be preserved through events. Thus, execution of two triggers leads to an infinite loop.

Table 6. INV PO of event trigger1 to check infinite loop

$\forall \text{id}.\text{id} \in \text{dom}(\text{empl_rec}) \wedge$ $\text{type} = \text{update} \wedge \text{table} = \text{BONUS} \vee$ $(\text{type} = \text{update} \wedge \text{table} = \text{EMPL})) \wedge$ $\text{emplid} \in \text{dom}(\text{bonus_rec})$ $\text{table} = \text{BONUS} \wedge f_bonus(\text{emplid}) > 10$ \vdash $\forall \text{id}.\text{id} \in \text{dom}(\{\text{emplid} \mapsto f_empl(\text{emplid}) + 1\} \oplus \text{empl_rec}) \wedge$ $\text{update} = \text{update} \wedge \text{EMPL} = \text{BONUS} \wedge$ $(\text{update} = \text{update} \wedge \text{EMPL} = \text{EMPL})$	trigger1 /INF_LOOP /INV
--	-------------------------------

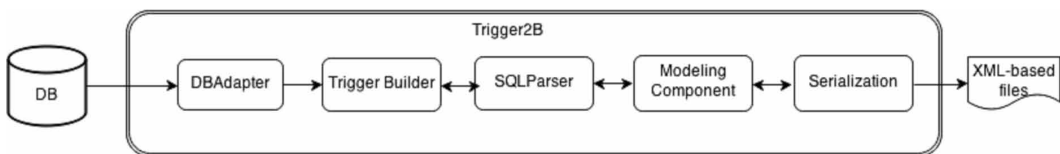
5. SUPPORT TOOL: TRIGGER2B

In this section, we present a tool named Trigger2B¹, which allows to translate automatically from database triggers to an Event-B model. First, we present architecture of the tool. After that, we show how main components of the tool are implemented.

5.1. Tool Architecture

Following the method presented in Section 3, we implement a tool called Trigger2B to support designing and modeling a database system including triggers. This tool can generate multiples XML-based format output which can be used later in verification phase with Event-B supporting tools such as RODIN platform. The architecture of this tool is illustrated in Figure 5.

Figure 5. Architecture of Trigger2Btool



Main components of the tool work as follows:

- **DBAdapter:** Connects to a relational database system to get information about the database which will be modeled such as existing tables and triggers;
- **Trigger Builder:** Allows users to create and design new triggers based on information of the connected database;
- **SQLParser:** Parses the content of all triggers designed within the connected database in order to extract necessary elements, e.g., type and table names of SQL statements, and forms trigger syntax tree;
- **Modeling Component:** Performs some algorithms to construct the target Event-B model from the trigger syntax tree;
- **Serialization:** Serialize the translated Event-B model to XML-based files such as RODIN/Event-B project files.

5.2. Implementation

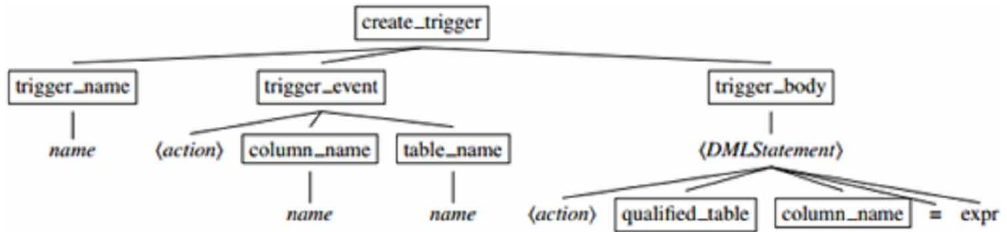
The heart of this tool is the modeling component which includes algorithms following the proposed translation rules to translate database concepts to Event-B constructs. The input of this component is the output of SQLParser component which currently uses ANTLR (2014) framework to parse sql statements. A parsed tree of a general trigger is partially illustrated in Figure 6.

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

We define a template based on form of StringTemplate, and then use the supporting RODIN API to serialize to Event-B components of the RODIN platform. A snippet code for defining RODIN project Event-B machines is defined as follows:

Figure 7(a) shows the RODIN output files which are generated automatically in the folder *res* by the tool after modeling the scenario described in Section 4.1. The output contains a full description of two triggers in Figure 7(b). An incomplete work is the invariant representing the data constraint because we have not defined it in the input SQL file yet. Therefore, we need to add this invariant

Figure 6. A partial parsed tree syntax of a general trigger



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

Input: Parsed syntax tree(t)
Output: Event-B model (M , C)

```

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 = table_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
  
```

clause and the theorem which is the disjunction of events' guards manually to obtain the complete model. After getting the complete model, proof obligations of infinite loops and data constraint properties are discharged automatically.

6. RELATED WORK

Many researches have been proposed for checking active rules or triggers. From the beginning, most of researchers have mainly focused on termination of triggers by using static analysis, e.g., checking set of triggers is acyclic with triggering graph. In Lee and Ling (1998, 1999), Sin-Yeung Lee and Tok-Wang introduced algorithms to detect correctness of updating triggers. However, this approach

Figure 7. The modeling result of the scenario generated by Trigger2B



is not extended apparently for general triggers and it is presented as their future work. Moreover, this approach also just can find the cycle of trigger execution but not proposed any formalization of the system. More recently, R. Manicka Chezian (2011) introduced a new algorithm, which does not pose any limitation on number of rules and only emphasizes algorithms detecting termination of the system. Though these results are powerful in detecting termination or cyclic properties of complex triggers, they lack formal representation.

Baralis (1999) improved existing techniques and proposed propagation algorithms to statically check if active rules are terminated or confluent. This approach, based on relational algebra, can be applied widely for active database rule languages and for trigger language (SQL:1999). However, the paper did not consider the constraint preservation properties and the approach seems too complex to be brought to practice.

Some work applied model checking techniques for active database rule analysis. Ghazi and Huth (1998) presented an abstract modeling framework for active database management systems and implemented a prototype of a Promela code generator. However, they did not describe how to model data and data actions for evaluation.

Choi et al. (2006a) proposed a general framework for modeling active database systems and rules. The authors address both termination and safety properties of active rules. Their work just focused on general active rules but not on exploring database triggers syntax in detail. The framework is feasible by using a model checking tool, e.g., SPIN, however, constructing a model in order to verify termination and safety properties is not a simple step.

Zang et al. (2008) proposed an approach to checking structural errors such as inconsistency, circularity, and redundancy of ECA rule-based systems. Their method classifies three different levels of verification and builds an EA tree to check each level. However, this approach cannot check

at some properties on the third level. Furthermore, they have not provided any supporting tool for building EA tree.

Recently, Ksystra et al. (2014) proposed a method to express and verify safety properties of reactive rules which also are ECA rules. It provides verification mechanism of termination, confluence and safety properties using CafeOBJ method.

Recently, Cacciagrano et al. (2018, 2020) proposed a set of techniques and tools for verifying redundancy, consistency, and usability of ECA rules based on a domain specific language for intelligent environment. But their work did not consider the correctness property of the rule.

Hamada Ibrahim et al (2020) proposed a framework for detecting and resolving conflicts of ECA rules based on SMT solvers. They also implemented a prototype tool using Java and web Service to evaluate their proposal.

In comparison with the prior work, our method is different and has some advantages. It recognizes the benefit of Event-B formal method when modeling database triggers because of the similarity between an ECA rule and Event-B event. Our translation rules handle triggers written in SQL. These rules also preserve data constraint preservation properties and infinite loops of the database system. The proposed method does not require any intermediate step to verify such properties. The tool Trigger2B supporting for partly automatic modeling makes our method feasible in practical software development.

7. CONCLUSION AND FUTURE WORK

Modeling and verification of database active rules or triggers are interesting topics for many research groups. Researchers have proposed many approaches with various techniques. Most of the existing work, however, focused on checking termination property, while the data constraint preservation has not been received much attention.

In this paper, we propose a new method to formalize and verify a database system including triggers with Event-B. We present a set of translation rules to encode the system in Event-B notations. Using Event-B for modeling the system is well-suited because ECA rules, which are used for describing behaviors of trigger systems, are matched to Event-B events. Therefore, modeling can be performed naturally. With the translation rules, the target model preserves data constraint properties. The proposed method can detect infinite loops of trigger execution by means of Event-B proof obligations at design phase. As far as we know, this paper reports the first concrete result of analyzing database triggers system.

Besides the advantages, this method still needs to be improved to model and verify more complex database systems with more complicated triggers. The current implementation is also limited to SQLite trigger syntax. Due to the limitation of the current RODIN/Event-B, we only can design each column of tables by the integer number type.

In the future, we will overcome the limitation of types in RODIN by incorporating Theory plugins Maamria (2013). We also intend to extend the tool to handle with more database systems, e.g., MySQL. In the theoretical perspective, we plan to extend the method by making use of Event-B composition to handle more complicated and nested triggers.

ACKNOWLEDGMENT

This work has been supported by VNU University of Engineering and Technology under Project QG.16.32.

REFERENCES

- ANTLR. (2014). <http://www.antlr.org>
- Abrial, J. R. (2010). *Modeling in Event-B: System and Software Engineering*. Cambridge University Press. doi:10.1017/CBO9781139195881
- Baralis, E. (1999). *Rule analysis. Active Rules in Database Systems*. Springer.
- Butler, M., & Maamria, I. (2013). Practical Theory Extension in Event-B. *Theories of Programming and Formal Methods, 8051*, 67–81. doi:10.1007/978-3-642-39698-4_5
- Cacciagrano, D. R., Corradini, F., Culmone, R., Gorogiannis, N., Mostarda, L., Raimondi, F., & Vannucchi, C. (2018). Analysis and Verification of ECA Rules in Intelligent Environments. *Journal of Ambient Intelligence and Smart Environments, 10*(3), 261–273. doi:10.3233/AIS-180487
- Cacciagrano & Culmone. (2020). IRON: Reliable domain specific language for programming IoT devices. *Internet of Things, 9*.
- Chavarría-Báez, L., & Li, X. (2007). Verification of active rule base via conditional colored petri nets. *SMC, 343* – 348. doi:10.1109/ICSMC.2007.4414028
- Choi, E. H., Tsuchiya, T., & Kikuno, T. (2006a). *Model checking active database rules. Technical report, AIST CVS*. Osaka University.
- Choi, E. H., Tsuchiya, T., & Kikuno, T. (2006b). Model checking active database rules under various rule processing strategies. *IPSJ Digital Courier, 2*, 826–839. doi:10.2197/ipsjdc.2.826
- Eisenberg, A., & Melton, J. (1999). Sql: 1999, formerly known as sql3. *SIGMOD Record, 28*(1), 131–138. doi:10.1145/309844.310075
- Ghazi, T., & Huth, M. (1998). *An Abstraction-Based Analysis of Rule Systems for Active Database Management Systems*. Technical report, Kansas State University. Technical Report KSU-CIS-98-6.
- Ibrahim, H., Khattab, S., Elsayed, K., Badr, A., & Nabil, E. (2020). A formal methods-based Rule Verification Framework for end-user programming in campus Building Automation System. *Building and Environment, 181*, 106983. doi:10.1016/j.buildenv.2020.106983
- Ksystra, K., Triantafyllou, N., & Stefaneas, P. (2014). On verifying reactive rules using rewriting logic. In A. Bikakis, P. Fodor, & D. Roman (Eds.), *Lecture Notes in Computer Science: Vol. 8620. Rules on the Web. From Theory to Applications* (pp. 67–81). Springer International Publishing. doi:10.1007/978-3-319-09870-8_5
- Le, H. A., & Truong, N. T. (2013). Modeling and Verifying DML Triggers Using event-B. In *Proceedings of the 5th Asian Conference on Intelligent Information and Database Systems* (vol. 2, pp. 539 – 548). Springer-Verlag. doi:10.1007/978-3-642-36543-0_55
- Lee, S. Y., & Ling, T. W. (1998). Are your trigger rules correct? In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*. IEEE Computer Society.
- Lee, S. Y., & Ling, T. W. (1999). Verify updating trigger correctness. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, (pp. 382 – 391). Springer-Verlag.
- Manicka, R., & Chezian, T. (2011). A New Algorithm to Detect the Non-Termination of Triggers in Active Databases. *International Journal of Advanced Networking and Applications, 3*(2), 1098–1104.
- Zhang, J., Moyne, J., & Tilbury, D. (2008). Verification of ECA rule-based management and control systems. In *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on* (pp. 1 – 7). doi:10.1109/COASE.2008.4626431

ENDNOTE

¹ The Trigger2B tool can be downloaded at: <http://uet.vnu.edu.vn/~thuantn/Trigger2B.zip>

Hong-Anh Le received Master Degree of Computer Science at Saarland University 2009. He received Ph.D. degree from VNU-University of Engineering and Technology in 2014. He currently is Dean of Faculty of Information Technology at Hanoi University of Mining and Geology, Vietnam. His research interests are formal methods, software testing, big data, AI in earth observation, climate changes, environment management, etc.

To Van Khanh (PhD) is a lecturer of Faculty of Information Technology, University of Engineering and Technology, Vietnam National University, Hanoi (VNU-UET). Dr. To Van Khanh is a computer scientist and obtained PhD degree in Japan in 2013. He started working at VNU-UET from 2004 as a lecturer assistant. Since 2013, he has been becoming full-time lecturer at Faculty of Information Technology. His courses are object - oriented programming, object – oriented analysis and design, fundamental of formal methods. Software verification, software testing, formal methods and back-end engines such as SAT and SMT solvers are his main research.

Ninh-Thuan Truong is an Associate Professor in Computer Science of VNU University of Engineering and Technology, Hanoi. He has graduated PhD in Computer Science in Nancy University, France in 2006. His research interests include software engineering, formal methods, software verification and testing.