Modeling and Verifying DML Triggers Using Event-B

Hong Anh Le[1] and Ninh Thuan Truong[2]

[1] Hanoi University of Mining and Geology
Dong Ngac, Tu Liem, Hanoi
[2] VNU - University of Engineering and Technology
144 Xuan Thuy, Cau Giay, Hanoi
{anhlh.di10,thuantn}@vnu.edu.vn

Abstract. Database trigger is a block code that automatically executes in response to changes of table or view in the database system. The correctness of a trigger usually can be verified when it is executed. It is apparently useful if we can detect the trigger system's errors in the design phase. In this paper, we introduce an approach to model and verify data manipulation language (DML) triggers in the database system by a formal method. In the first phase, we formalize a database trigger system by an Event-B model. After that, we use the Rodin tool to verify some properties of the system such as termination, preservation of constraint rules. We also run an example to illustrate the approach in detail.

1 Introduction

Triggers are active rules of some commercial database systems such as Oracle, SyBase, etc.. which are formed in ECA (Event - Condition - Action) structure. Triggers are widely and commonly used in database systems of many applications to implement automatic tasks and ensure integrity constraints. In some commercial databases, triggers have two kinds: data manipulation language (DML) and system triggers. The former are fired whenever events such as DELETING, UPDATING, INSERTING occur, while the latter are executed in case that system or data definition language (DDL) events occur. A trigger is made of a block of code, for example in Oracle, a trigger is similar to a stored procedure containing blocks of PL/SQL code. These codes are human readable and without any formal semantic. Therefore, we can only verify that if a trigger terminates or conflicts to integrity constraints after executing it or with human inspection step by step. It is important if we can show that triggers execution is correct at the design time. Several works have attempted to solve this question by using termination detection algorithms or model checking [13] [14] [10] [7] [16]. However, in our thought, most of results focused on the termination property, while few of them addressed to both termination and integrity constraints of the database system. Furthermore, these approaches seem such complicated that we can not apply them in the database development.

A. Selamat et al. (Eds.): ACIIDS 2013, Part II, LNAI 7803, pp. 539–548, 2013.

The B method [1] is a formal software development method, originally created by J.-R. Abrial. The B notations are based on the set theory, generalized substitutions and the first order logic. Event-B [2] is an evolution of the B method that is more suitable for developing large reactive and distributed systems. Software development in Event-B begins by abstractly specifying the requirements of the whole system and then refining them through several steps to reach a description of the system in such a detail that can be translated into code. The consistency of each model and the relationship between an abstract model and its refinements are obtained by formal proofs. Support tools have been provided for Event-B specification and proof in the Rodin platform.

In this paper, we propose an approach to formalize database triggers system by a proof-based method, e.g Event-B. The main idea of the approach comes from the similarity between structures of Event-B EVENT and ECA. We first translate a database system to an Event-B model. In the next step, we bring this model to more practical approach by using the Rodin platform to verify some properties such as termination and constraint preservation based on its proof obligation engine. The advantage of our approach is that a real database system including triggers and constraints can be modeled easily by logic expression phrases in Event-B such as INVARIANTS and EVENTS. Therefore, the correctness of the entire system can be achieved by formal proofs. It is valuable especially for database developers since they are able to ensure that the trigger systems avoid the critical issues at the design time. Furthermore, the approach is such practical that we can implement a tool following its main idea to transform a database model to an Event-B model in Rodin platform automatically (or partly). It makes sense as we can bring the formal verification to database implementation. It also overcomes one of disadvantages that makes formal methods absent in the database development process because of the complexity of modeling.

The remainder of this paper is structured as follows. Section 2 gives a brief introduction of Event-B and background of database triggers. Next, in Section 3, we introduce some transformation rules between a database triggers system to an Event-B model. To show the approach in detail, we model a specific trigger system in an example in Section 4. Followed by Section 5, we give some information and adjustment of related works so far. We conclude our contribution and present the future works in Section 6.

2 Backgrounds

In this section, we briefly introduce the overview of relational database triggers and basic knowledge of Event-B.

2.1 Database Triggers

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 is followed EAC structure, hence it takes the following form: *rule_name:: Event(e) IF condition DO action.*

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

2.2 Event-B

Event-B is a kind of formal method which combines mathematical techniques from the set theory and the first order logic. It is used as a notation and method for the formal development of discrete systems. Event- B is an evolution of others formal method notations like B-method (also know as classical B), Z and Action Systems. It is considered as an evolution because it simplifies the B machine notations, is easy to learn and more suitable for parallel and distributed reactive system development. Another advantage Event-B is the tool support for system modeling. The basic structure of an Event B model consists of a MACHINE and a CONTEXT.

Contexts form the static part of the model while machines form the dynamic part. Contexts can extend (or be extended by) other context and are referred (seen) by machines. The machine contains the dynamic part of the model. It describes the system state, the operations to interact with the environment together with the properties, conditions and constraints on the model. A Machine is defined by a set of clauses which is able to refine another Machine. We briefly introduce main concepts in Event-B as follows:

- Variables: represents the state variables of the model of the specification.
- Invariants: describes by first order logic expressions, the properties of the attributes defined in the variable clauses. Typing information, functional and safety properties are described in this clause. These properties are true in the whole model. Invariants need to be preserved by events clauses.
- Events: defines all the events that occur in a given model. Each event is characterized by its guard (i.e. a first order logic expression involving variables). An event is fired when its guard evaluates to true. If several guards evaluate to true, only one is fired with a non deterministic choice.

A Context consists of the following items:

- Sets: describes a set of abstract and enumerated types.
- Constants: represents the constants used by the model.

– Axioms: describes with first order logic expressions, the properties of the attributes defined in the CONSTANTS clause. Types and constraints are described in this clause.

After having the system modeled in Event-B, we need to reason about the model to understand it. To reason about a model, we use its proof obligation which show its soundness or verify some properties. As we mention in the first part of this Subsection, behaviors of the system are represented by machines. Variables v of a machine defines state of a machine which are constrained by invariants $I(v)$. Events E_m which describe possible changes of state consisting of guards $G_m(v)$ and actions $S_m(v, v')$ and they are denoted by

$$\text{when } G_m(v) \text{ then } v :| \ S_m(v, v') \text{ end}$$

Properties of an Event-B model are proved by using proof obligations (PO) which are generated automatically by the proof obligation generator of Rodin platform. The outcome of the proof obligation generator are transmitted to the prover of the Rodin tool performing automatic or interactive proofs.

3 Modeling and Verifying Database Triggers System

In this section, we present an approach to model a database systems including triggers. The main idea is mapping between entities of the database systems and Event-B elements in which we emphasize on modeling triggers by Event-B Events. After the transformation, we are able to verify some properties based on achieved Event-B model.

3.1 Modeling Database Systems

A database system is normally designed by several elements such as tables (or views) with integrity constraints and triggers. Whenever users modify the database table contents, e.g Insert, Delete and Update actions, the modification should be conformed to constraints and it also can fire the corresponding triggers. Before modeling the trigger system by Event-B, we introduce some definitions related to Event-B specification, they are useful in the modeling process.

Definition 1. A database trigger is modeled by a 3-tuple $db = <T, C, G>$ where T is a set of table, C states system constraints, G indicates a set of triggers.

Definition 2. For each $t \in T$, denoted by a tuple $t = <row_1, .., row_m>$where m is the number of table row, row_i, $(i \in 1..m)$ is a set indicating the i-th row of the table. A row is stated by a tuple $row_i = <field_1, .., field_n>$

Definition 3. Each trigger g of the system is presented as a 3-tuple such as $g \in G, g = <e, c, a>$ where, e is the corresponding event of the trigger, c is condition of the trigger, a is the action of the trigger.

We model a database system by mapping these definitions to Event-B concepts in Table 1. These rules are described in detail as follows:

Table 1. Transformation between database system and Event-B concepts

	Database definitions	**Event-B concepts**
Rule 1.	$db = <T, C, G>$ $T = \{t_1, .., t_m>$	$db_B = \{S_T \leftrightarrow I \leftrightarrow E\}$ $S_T = \{t_1, .., t_m\}$
Rule 2	$t = <r_1, .., r_m>$	$t_B = \{r_{B1}, .., r_{Bm}\}$.
Rule 3	$r_i = <f_{i1}, .., f_{in}>$	$r_{Bi} = \{1 \mapsto f_{Bi1}, .., m \mapsto f_{Bin}\}$

- Rule 1. Where set of tables T is mapped to set S_T, constraints C is translated to a set of invariant I, triggers set G is transformed to a set of events E
- Rule 2. A table is translated to a set of rows.
- Rule 3. A row of a table is transformed to an ordered set of fields, where m is a number of columns of the table and f_{Bij} is the value of column j at row i, where $i \in 1..m, j \in 1..n$

In the next subsection, we present in detail how to formalize database triggers.

3.2 Formalizing Triggers

As illustrated in Table 2, a trigger is formalized as an Event-B event where trigger's type and its condition is the guard of the event. Action of a trigger is transformed to the body part of an Event-B event.

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

IF (*type*) ON (*condition*)	WHEN (*type* $\wedge$ *condition*)
ACTION (act)	THEN (act) END

To show our approach, we simplify by considering the case that the Action part of a trigger contains a single action, though it can be a sequence of actions. It is clear that we are able to model such sequence of actions using Event-B if we can formalize a single Action. An Action of a trigger body is Insert, Update or Delete statement. In case of Update and Delete statements, the action contains a condition that shows which rows are affected. Therefore, we combine statement and trigger condition into guard of transformed event. Specifically, mapping rules of each kind of statements are presented in Table 3.

3.3 Verifying System Properties

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

Table 3. Translating SQL statements to Event-B events

UPDATE table_name SET column1=value, column2=value2 WHERE some_column=some_value	WHEN *update_condition* THEN $r := \{1 \mapsto value1, 2 \mapsto value2\}$
DELETE FROM table_name WHERE some_column=some_value	WHEN *delete_condition* $table\_name := table\_name - \{col_1 \mapsto val_1, .., col_n \mapsto val_n\}$
INSERT INTO table_name VALUES (value1,..,valuen)	WHEN *insert_condition* $table\_name := table\_name \cup \{col_1 \mapsto val_1, .., col_n \mapsto val_n\}$

- Termination: Since a trigger can fire the other triggers, hence it probably leads to infinite loop. The termination of a trigger is able to be verified by the deadlock property of the Event-B model. This situation occurs when after a sequence of events, state of the system does change. This property is proved by proof obligations which state that the disjunction of the event guards always hold under the properties of the constant and the invariant. The deadlock freedom rule is stated as $I(v) \vdash G_1(v) \vee ... \vee G_n(v)$, where v is variable, $I(v)$ denotes invariant, $G_i(v)$ presents guard of the event. At the moment, the deadlock freeness PO is not generated automatically by the Rodin tool yet. However, we can generate it ourself by as a theorem saying the disjunction of guards.
- Constraint preservation: Since these properties already are modeled by Event-B INVARIANTS as the approach illustrated in Subsection 3.1, hence we can prove them by using invariant PO rules.

4 An Example

In order to make our approach more clear, in this section, we take an example to present it in detail. We first describe the example, after that we model it by an Event-B machine and verify its properties.

4.1 Example Description

Let assume that we have a database system including two tables EMPLOYEES and BONUS structured in Table 4.

Table 4. Table EMPLOYEES and BONUS

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

The database system has a constraint: *The bonus of an employee with a level greater than 5 is at least 20.*

It includes two triggers that do the following tasks:
Trigger 1. Whenever the level of employee is updated, his bonus is increased by 10 if the level is even
Trigger 2. If the employee's bonus amount is updated, then his level is increased by 1.

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

```
CREATE TRIGGER Trigger_1 BEFORE UPDATE
    OF level ON employees
    FOR EACH ROW
    BEGIN
          IF MOD(employees.level,2)=0 THEN
             UPDATE bonus SET bonus.amount
                    =bonus.amount  + 10
       WHERE bonus.E_id = employees.E_id;
    END IF;
END
```

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

4.2 Modeling an Example

Followed the approach presented in Section 3, we formalize two tables which are involved in the trigger statements by two variables such as *empl* and *bonus*. Variables *bonus_rec* and *empl_rec* present a row of the table Bonus and Employees respectively.

$\texttt{inv7}: \texttt{bonus} \in \mathbb{P}((\mathbb{N}_1 \times \mathbb{N}_1) \times (\mathbb{N}_1 \times \mathbb{N}_1))$
$\texttt{inv11}: \texttt{empl} \in \mathbb{P}((\mathbb{N}_1 \times \mathbb{N}_1) \times (\mathbb{N}_1 \times \mathbb{N}_1))$
$\texttt{inv16}: \texttt{bonus\_rec} \in \texttt{bonus}$
$\texttt{inv17}: \texttt{empl\_rec} \in \texttt{empl}$
$\texttt{inv5}: \texttt{trigger\_type} \in \mathbb{P}(\texttt{TIME}) \leftrightarrow \mathbb{P}(\texttt{COMMAND})$
$\texttt{inv20}: \texttt{active\_field} \in \mathbb{P}(\texttt{TABLE\_NAMES}) \leftrightarrow \mathbb{P}(\texttt{FIELD\_NAMES})$

The constraint of the database system is also formalized by an INVARIANT

INVARIANTS
$\texttt{inv21}: \texttt{empl\_level} < 5 \lor \texttt{bonus\_amount} \geq 20$

We next formalize two triggers of the system as the approach presented in 3.2. Since DML actions are performed on the table, we model the table involved in triggers by an Event-B VARIABLE *table* such that *table* is the identifier of the table.

Event *trigger1* $\widehat{=}$
  when
    grd1 : $\texttt{trigger\_type} = \{\texttt{AFTER} \mapsto \texttt{UPDATE}\}$
    grd3 : $\texttt{empl\_levelmod2} = 0$
    grd5 : $\texttt{empl\_level} \in \texttt{ran}(\texttt{ran}(\texttt{empl}))$
    grd6 : $\texttt{empl\_id} \in \texttt{ran}(\texttt{dom}(\texttt{empl}))$
    grd8 : $\texttt{ran}(\texttt{dom}(\{\texttt{bonus\_rec}\})) = \{\texttt{empl\_id}\}$
    grd9 : $\texttt{bonus\_amount} \in \texttt{dom}(\texttt{dom}(\{\texttt{bonus\_rec}\}))$
    grd10 : $\texttt{active\_field} = \{\texttt{EMPLOYEES} \mapsto \texttt{EMP\_LEVEL}\}$
  then
    act1 : $\texttt{trigger\_type} := \{\texttt{AFTER} \mapsto \texttt{UPDATE}\}$
    act3 : $\texttt{bonus\_amount} := \texttt{bonus\_amount} + 10$
    act5 : $\texttt{bonus\_rec} := (1 \mapsto \texttt{empl\_id}) \mapsto (2 \mapsto \texttt{bonus\_amount})$
    act6 : $\texttt{active\_field} := \{\texttt{BONUS} \mapsto \texttt{BONUS\_AMOUNT}\}$
  end
Event *trigger2* $\widehat{=}$
  when
    grd1 : $\texttt{trigger\_type} = \{\texttt{AFTER} \mapsto \texttt{UPDATE}\}$
    grd2 : $\texttt{active\_field} = \{\texttt{BONUS} \mapsto \texttt{BONUS\_AMOUNT}\}$
    grd3 : $\texttt{ran}(\texttt{ran}(\{\texttt{empl\_rec}\})) = \{\texttt{empl\_id}\}$
    grd4 : $\texttt{empl\_level} \in \texttt{ran}(\texttt{ran}(\{\texttt{empl\_rec}\}))$
  then
    act1 : $\texttt{trigger\_type} := \{\texttt{AFTER} \mapsto \texttt{UPDATE}\}$
    act2 : $\texttt{empl\_level} := \texttt{empl\_level} + 1$
    act3 : $\texttt{empl\_rec} := (1 \mapsto \texttt{empl\_id}) \mapsto (2 \mapsto \texttt{empl\_level})$
    act4 : $\texttt{active\_field} := \{\texttt{EMPLOYEES} \mapsto \texttt{EMP\_LEVEL}\}$
  end

4.3 Checking Properties

- Termination: To verify the termination property in the Rodin tool, we generate an invariant clause which is the disjuntion of two events' guards. Using PO engine of the Rodin tool, we can prove that the system is not deadlock free, i.e the system is terminated.
- Constraint violation: Since the constraint property of the system is modeled by INVARIANT *inv*21, hence it is also proved by invariant preservation rules. The invariant is proved to be failed through events of the model, hence the triggers execution violates the system constraint.

5 Related Works

From the beginning, the previous works focused on the termination of the triggers by using static analysis, e.g. checking set of triggers is acyclic with triggering

graph. In [13] and [14], Sin-Yeung Lee and Tok-Wang introduced algorithms to detect the correctness of updating triggers. However, this approach is not extended apparently for general triggers and it is presented as their future work.

E.Baralis *et al* performed the dynamic analysis to check active rules at run time to see if a state of the database system is repeated.

L. Chavarria and Xiaoou Li proposed a method verifying active rules by using conditional colored Petri nets [7]. Since Petri nets are mainly used in modeling transitions, hence it is quite elaborated when normalizing rules. The approach has to classify rules by the logic condition of these rules to check if they involve disjunction or conjunction operators. In our opinion, if the number of these operators are enormous then the transition states can be exploded.

Some works applied model checking for active database rule analysis [12][10]. In [12], T. S. Ghazi and M. Huth presented an abstract modeling framework for active database management systems and implemented a prototype of a Promela code generator. However, they did not describe how to model data and data actions for evaluation.

Eun-Hye CHOI *et al* [10] proposed a general framework for modeling active database systems and rules. The framework is feasible by using a model checking tool, e.g SPIN, however, constructing a model in order to verify the termination and safety properties is not a simple step and can not be done automatically.

More recently, R. Manicka Chezian and T.Devi [17] introduced a new algorithm which does not pose any limitation on the number of rules but it only emphasizes on algorithms detecting the termination of the system.

6 Conclusion and Future Works

Most of the researches to date that have worked on verifying and modeling database active rules or triggers mainly focuses on the termination property. A few works presented methods to model a database system and verify some properties of the system. However, in our opinion, these results are complex to bring them to software development and are not feasible to be performed automatically without human analysis. In this paper, we propose an approach to formalize and verify the database system with constraints and triggers by using Event-B. Our main contribution is that we perform the mapping between elements of the database such as tables, triggers, constraints to Event-B clauses. We also reuse the obligation engines and tool supported by Event-B to prove the correctness of the system. Moreover, the transformation is also simple and clear such that it is feasible to formalize the database system by an Event-B model automatically. Therefore, it makes sense if we want to bring the formal verification to software development.

Besides the advantages, the paper still has some limitation such that we do not address how to model a more complex case study with more complicated triggers. These issues, along with development of a tool which takes into account to translate a database system to an Event-B model in the format of Rodin platform, are our future works.

Acknowledgments. This work is supported by the project no. 102.02–2010.06 granted by Vietnam National Foundation for Science and Technology Development (Nafosted).

References

1. B method web site (2012), `http://www.bmethod.com`
2. Event-b and the rodin platform (2012), `http://www.event-b.org`
3. Abrial, J.-R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
4. Ait-Sadoune, I., Ait-Ameur, Y.: From bpel to event-b. In: IM FMT 2009 Conference, Dsseldorf Germany, Fevruary (2009)
5. Baralis, E.: Rule analysis. In: Active Rules in Database Systems, pp. 51–67. Springer, New York (1999)
6. Baralis, E., Widom, J.: An algebraic approach to static analysis of active database rules. ACM Trans. Database Syst. 25(3), 269–332 (2000)
7. Chavarría-Báez, L., Li, X.: Verification of active rule base via conditional colored petri nets. In: SMC, pp. 343–348 (2007)
8. Chavarría-Báez, L., Li, X.: Ecapnver: A software tool to verify active rule bases. In: ICTAI (2), pp. 138–141 (2010)
9. Chavarría-Báez, L., Li, X.: A petri net-based metric for active rule validation. In: ICTAI, pp. 922–923 (2011)
10. Choi, E.-H., Tsuchiya, T., Kikuno, T.: Model checking active database rules. Technical report, AIST CVS, Osaka University, Japan (2006)
11. Choi, E.-H., Tsuchiya, T., Kikuno, T.: Model checking active database rules under various rule processing strategies. IPSJ Digital Courier 2, 826–839 (2006)
12. Ghazi, T., Huth, M.: An Abstraction-Based Analysis of Rule Systems for Active Database Management Systems. Technical report, Kansas State University, Technical Report KSU-CIS-98-6, p.15 (April 1998)
13. Lee, S.-Y., Ling, T.-W.: Are your trigger rules correct? In: Proceedings of the 9th International Workshop on Database and Expert Systems Applications, DEXA 1998, p. 837. IEEE Computer Society, Washington, DC (1998)
14. Lee, S.-Y., Ling, T.-W.: Verify Updating Trigger Correctness. In: Bench-Capon, T.J.M., Soda, G., Tjoa, A.M. (eds.) DEXA 1999. LNCS, vol. 1677, pp. 382–391. Springer, Heidelberg (1999)
15. Li, X., Medina Marín, J., Chapa, S.V.: A Structural Model of ECA Rules in Active Database. In: Coello Coello, C.A., de Albornoz, Á., Sucar, L.E., Battistutti, O.C. (eds.) MICAI 2002. LNCS (LNAI), vol. 2313, pp. 486–493. Springer, Heidelberg (2002)
16. Ray, I., Ray, I.: Detecting termination of active database rules using symbolic model checking. In: Caplinskas, A., Eder, J. (eds.) ADBIS 2001. LNCS, vol. 2151, pp. 266–279. Springer, Heidelberg (2001)
17. Manicka chezian, T.R.: A new algorithm to detect the non-termination of triggers in active databases. International Journal of Advanced Networking and Applications 3(2), 1098–1104 (2011)