Using UML's Sequence Diagrams for Representing Execution Models Associated to Triggers*

Harith T. Al-Jumaily, César de Pablo, Dolores Cuadra, and Paloma Martínez

Computer Science Department, Universidad Carlos III de Madrid
{haljumai, cdepablo, dcuadra, pmf}@inf.uc3m.es

Abstract. Using active rules or triggers to verify integrity constraints is a serious and complex problem because these mechanisms have behaviour that could be difficult to predict in a complex database. The situation is even worse as there are few tools available for developing and verifying them. We believe that automatic support for trigger development and verification would help database developers to adopt triggers in the database design process. Therefore, in this work we suggest a visualization add-in tool that represents and verifies triggers execution by using UML's sequence diagrams. This tool is added in RATIONAL ROSE and it simulates the execution sequence of a set of triggers when a DML operation is produced. This tool uses the SQL standard to express the triggers semantics and execution.

1 Introduction

The passive behaviour of traditional databases often causes semantic loss in database systems. Users and applications always have the responsibility to protect these semantics. For this reason, traditional databases have been improved by adopting active behaviour. An active behaviour is a complex operation that is activated in an autonomous way to perform predefined actions. Usually, this behaviour is known as triggers or ECA rules. An ECA rule consists of three components; event, condition, and action. The execution model of ECA rules follows a sequence of steps: event detection, condition test, and action execution. An event in relational databases is a DML (Data Manipulation Language) statement such as (INSERT, DELETE, and UPDATE). Once a trigger is activated and its condition is evaluated to true, the predefined actions are automatically executed.

Incorporating active rules enhances the functionality of database systems and provides flexible alternatives to implement many database features, such as to enforce integrity constraints [1]. Because of execution models of triggers, an active database is more complicated than a passive one. For that reason we believe that automatic support for triggers development could help to adopt active rules by database designers and developers. The validation of active rules/triggers execution is the major problem that makes the application development a difficult task. As rules can act in a way that leads to conflict and undesirable problems, the developer needs

* This work is part of the project *"Software Process Management Platform: modelling, reuse and measurement"*. TIN2004/07083.

additional effort to control this behaviour. The objective of this validation is to guarantee the successful execution of the triggers; that means to avoid non-termination state in their execution.

In commercial CASE tools which support triggers development, we have detected that developing triggers and plugging them into a given model is an insufficient task because of the behaviour of such triggers is invisible to the developers.

Therefore, in this work we suggest a visualization tool to represent and verify triggers execution by using UML's sequence diagrams. This tool has three contributions. First, we use the SQL standard [5] to express triggers semantics and execution. Second, we use the UML sequence diagram to display interactions between triggers. And finally, we use a commercial CASE tool (Rational Rose) to add-in our approach. These contributions make our approach quite useful, practical and intuitive to manage triggers using the triggering graph (TG) for checking non termination state. TG is one of the most important tools in active rules to check the termination execution for a set of rules and we adopt it for our approach.

The rest of this work is organised as follows. In section (2) the semantics of triggers execution is explained according to the SQL standard. In section (3) works related to rules behaviour analyzer tools or visualization tools are presented. In section (4) we will explain our visualization tool design. Finally, in section (5) some conclusions and future works are exposed.

2 Triggers Execution in SQl3

This section addresses common components according to the definition specified in the SQL2003 standard which makes revisions to all parts of SQL99 and adds new features [4]. A SQL standard trigger is a named event-condition-action rule that is activated by a database state transition. Every trigger is associated with a table and is activated whenever that table is modified. Once a trigger is activated and its condition evaluated to true, the trigger's action is performed. When we talk about the semantics of triggers execution in the SQL standard, we consider the Knowledge Model and the Execution Model.

A knowledge model supports the description of the active functionality; it is considered to have three principal components; an event, a condition, and an action [1]. The SQL3 syntax of triggers is shown in Fig.1.

In database systems, the triggers execution model specifies how a set of triggers is treated and executed at runtime. Although triggers are available in most DBMS, unfortunately their execution models change from one DBMS to another. Despite this fact, a common execution strategy is shared among systems according to the two main requirements for the SQL triggers execution. These requirements are [1]; (1) the execution model must ensure consistency in the database, and (2) all Before-triggers and After-triggers must be execute before or after the associated table will be modified, respectively. Before-triggers are especially useful for maintaining a constraint at the time that the data changes, while After-triggers are useful for invoking functions and stored procedures which execute modification operations inside and outside the database.

```
CREATE TRIGGER <trigger name>
[BEFORE | AFTER]
[INSERT | DELETE | UPDATE [OF <trigger column name>]]
ON <table name>
[REFERENCING <Old | New [Row | Table]>]
[FOR EACH {ROW | STATEMENT}]
[WHEN < condition >]
BEGIN ATOMIC
{< SQL procedure statement…… >}...
END;
```

Fig. 1. The SQL standard triggers syntax

The SQL standard allows the definition of multiple triggers associated to the same table, same event, and same activation time. Multiple triggers can simultaneously be selected for the execution. When multiple triggers are activated at the same time, priorities are used to resolve triggers executions. A trigger with the highest priority is executed first [5].

3 Related Work

As many works have been done in the area of static analysis [17] [18], we use in our analyzer the concept of Triggering Graph (TG) to detect non-termination states. TG is a straightforward graph where each node T_i corresponds to a rule and a direct arc between T_1 and T_2 is the event which belongs to T_1's action and causes the activation of T_2. A cycle is produced in TG when a rule T_i may trigger itself or when T_i triggers the same initial subset. In the figure (2), the subset of active rules $S=\{T_1, T_2, T_3\}$ is a cycle when the rule T_1 is fired again by the event e_3. The termination analysis itself focuses on identifying and eliminating arcs that could introduce cycles into the TG [19]. Redefining the rule T_3 and reconstructing again the graph TG is a good solution to verify the termination state of the subset S.

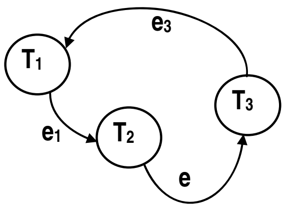

Fig. 2. The TG of the rules execution

Rules behaviour analyzer tools or visualization tools have been received strong interest from active database community where various works have been mentioned in the literature on using these tools in the verification of triggers execution. Arachne [10] is one of such tools; it is used in the context of object oriented (OO) active database systems. It accepts as input a set of Chimera active rules, and it uses triggering graph analysis to detect non-termination behaviour at compile-time. Active rules terminate if the triggering graph is acyclic. Once a cycle is detected, the user is responsible for assuring the termination. VITAL [11] is another set of development tools; it includes a tool for static analysis. This tool uses the triggering graph for detecting cycles in a set of active rules. TriGS [12] is a graphical development tool of active OO database application, it has been specifically designed for Trigger system for GemSione. DEAR [13] tool has been implemented on an active OO database system. It mainly focuses on a display form to rules interaction and sends information to users to help them to detect errors.

On the other hand, multiple efforts have been devoted to face database modelling problems. One of these problems is the automatization of database design process using CASE tools. Frequently, these tools do not completely support all phases of analysis and design methodology of databases. Triggers development is supported by some of these CASE tools such as Rational Rose [14], ERwin [15], Designer2000 [16]. These tools provide editors to allow users define different types of triggers. Furthermore, ERwin allows users generate triggers which enforce integrity constraints. The drawback of these CASE tools is that the termination of triggers execution is not guaranteed. Until now, there is no way to allow users of these tools to verify the developed triggers without need to execute them in a real database.

4 Visualization Tool Design

For explaining our approach, let us consider the example shown in figure 3 (a). It is a simple database schema using UML class diagram. It has three persistent classes (Student, Professor, and Department). The mapping of each class and association into Rational Rose Data model [20] [21] is shown in figure 3(b), and the data model transformation into relational model is shown in figure 3(a).

The main objective of our tool is to display triggers execution scenarios and to send messages to users for indicating whether these scenarios terminate or it is necessary the users' intervention to resolve a non termination execution. In this proposal, we show triggers and integrity constraints interaction in a display form as well as the non termination problem.

On the other hand, the UML's sequence diagram is used to show the interactions between objects and events in a sequential order according to the time. It is a two-dimension diagram, the vertical dimension is the time axis, and the horizontal dimension shows objects roles in their interactions.

In the context of triggers execution, it is very helpful to employ a tool to show the behaviour and the interactions of triggers that belong to a model. Therefore, we will use the sequence diagram elements to interpret the execution of triggers associated to

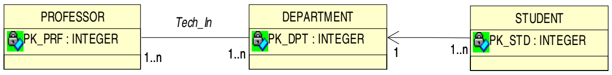

Fig. 3(a). Class diagram

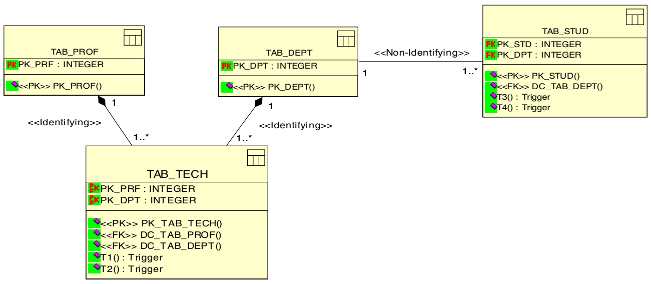

Fig. 3(b). Transformation of (a) into Rational Rose Data model

```
Create Table TAB_DEPT (PK_DPT Primary Key …);
Create Table TAB_STUD (PK_STD Primary Key, PK_DPT,
Constraint DC_TAB_DEPT References TAB_DEPT(PK_DPT)
On Delete Cascade);
Create Table TAB_PROF (PK_PRF Primary Key …);
Create Table TAB_TECH (PK_PRF, PK_DPT,
Constraint DC_TAB_PROF FOREIGN KEY (PK_PRF)
References TAB_PROF(PK_PRF) On Delete Cascade,
Constraint DC_TAB_DEPT FOREIGN KEY (PK_DPT)
References TAB_DEPT(PK_DPT) On Delete Cascade);
```

Fig. 3(c). Transformation of (b) into relational model

a relational schema. We use Rational Rose CASE tool to implement our approach because it is able to easily add-in software tools. It can be accessed from the Tools menu.

4.1 Used UML Notation

In this section, we will explain how we use the UML notation [22] to represent triggers execution and how we apply sequence diagrams to detect the non termination problem. The figure (4) shows an example of a sequence diagram.

- **Scenario Diagram:** A scenario is an instance of a use case that describes the sequential occurrences of events during the system execution. Sequence diagrams

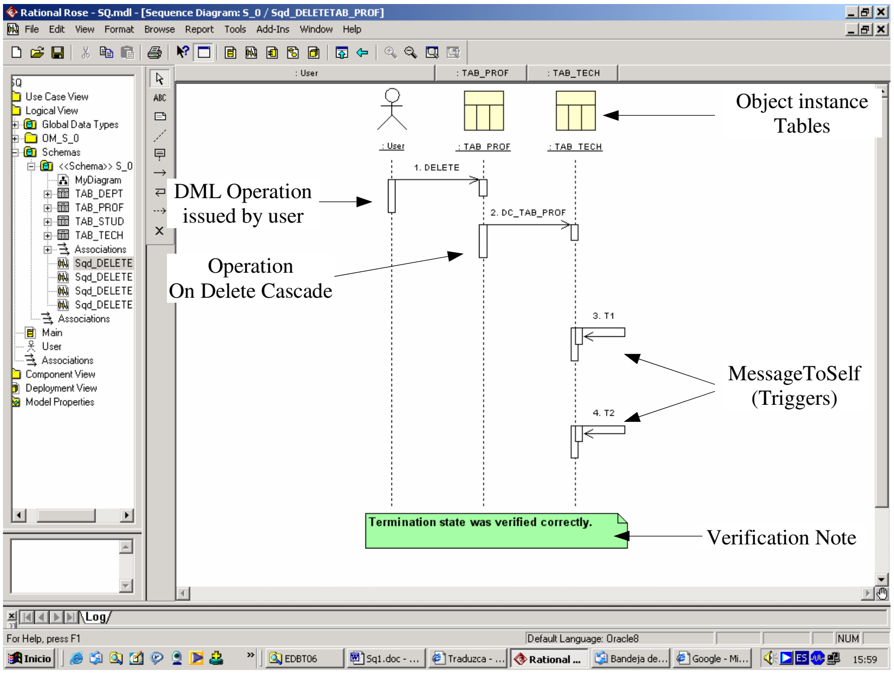

Fig. 4. A Scenario Diagram (Rational Rose)

allow users to create a display form of a scenario. In our approach, we create a scenario diagram for each event that may be generated on a table and the sequence of events and operations that follow after that event. Therefore, for each object table in the model, we need to create three scenario diagrams, one for each DML statement (INSERT, DELETE, and UPDATE).

- **Tables:** Tables are represented in Rational Rose as a stereotype of an object instance. The scenario diagram contains one or more object instances which have behaviour to be shown in the diagram. A table has three basic behaviours relevant for static analysis of the termination which are the three DML operations (INSERT, UPDATE, and DELETE). An object instance has a lifeline which represents the existence of the object over a period of time.
- **Message:** Messages in a sequence diagram are methods or operations which are used to illustrate the object behaviour. A message is the communication carried between two objects to define the interaction between them. A message is represented in the sequence diagram by using the message icon connecting two lifelines together. The message icons appear as solid arrows with a sequence number and a message label. The first message always starts at the top of the diagram and others messages follow it. When theSender=theReceiver, this means that the object theSender is sending a message to itself, MessageToSelf. Each message is associated with an integer number that shows the relative position of the message in the diagram. For example, if theSequence=3, the message is the third message in the diagram.

- **Note:** We use notes to warn users about the results of the verification. Our tool represents two types of notes to the users. The first is "*Termination state was correctly verified*" which is sent when the execution of a given scenario is correctly terminated. The second note is "*Non termination state was detected. Please, solve the problem and try again*". This note is sent when the verification of the scenario detects a non termination state in the execution of triggers.

4.2 Applying Sequence Diagrams

In general, triggers which are associated to a table are activated when that table is modified. In this context, when a trigger is fired it must examine the associated table and all other tables that can be modified by it. When an activated trigger examines its associated table, this is exactly like when an object sends a message to itself theSender=theReceiver. Therefore, a trigger instance is represented in sequence diagrams as MessageToSelf (figure 4). The trigger name is included into the message icon. BEFORE-triggers and AFTER-triggers are represented by using the same notation MessageToSelf. Trigger conditions are not considered because we use the static analysis approach to detect non termination state.

On the other hand, we used the notation Message for the other operations related to the behaviour. Such operations are shown below:

- The first operation that starts the scenario diagram. This message represents the operation which is sent from the user to a given object to start the scenario.
- DML statements (INSERT, DELETE, and UPDATE) included in a trigger's action and modify other object table theSender≠theReceiver.
- Referential constraint actions, CASCADE, SET DEFULT, and SET NULL are considered. We represent these actions in the sequence diagrams as messages from the parent object to the child object. The name and the type of the operation are indicated on the message icon.

The message icon used to represent these operations is a solid horizontal arrow with a sequence number and a message label.

The most important aspects that distinguish the SQL standard trigger execution model from others models such as, (Ariel [6], HiPac [7], and Starburst [8]) are the interactions between the triggers and the referential constraint actions [9]. In relational databases, the tables are represented by sets of rows and connections between tables are represented by defining foreign keys. Referential integrity constraints are predicates on a database state that must be evaluated, if these restrictions are violated the database is in an inconsistent state. In order to maintain the referential integrity of the database, the SQL standard uses actions such as NO ACTION, RESTRICT, CASCADE, SET DEFAULT, and SET NULL. In this work, we consider the last three actions because they produce interactions among triggers.

We will present in this section two scenarios to illustrate the usefulness of our tool.

Scenario 1
Let us consider that the table TAB_TECH has two triggers (figure 3(b)). The descriptions of these triggers are shown as follows:

```
CREATE TRIGGER T1                CREATE TRIGGER T2
AFTER DELETE ON TAB_TECH         AFTER DELETE ON TAB_TECH
WHEN C1                          WHEN C2
BEGIN ATOMIC                     BEGIN ATOMIC
X:=1;                            Y:=X;
END;                             END;
```

The scenario 1 (figure 4) starts when the user **Actor** carries out the operation (1: DELETE) in the table TAB_PROF. This table does not have any associated trigger, but when this operation is carried out, the referential action On Delete Cascade (2:DC_TAB_PROF) in TAB_TECH is executed; then the two triggers (3: T1) and (4: T2) are executed as well. As figure (4) shows, when the execution of T2 is finished the transaction is ended, so termination state is reached. The execution sequence of this scenario is shown below:

1: When DELETE FROM TAB_PROF is carried out ⇒
2: The DC_TAB_PROF is executed (section 2.3) ⇒
3: T1 (X=1) is executed ⇒
4: T2 (Y=X) is executed ⇒ END. "*Termination state was correctly verified*"

Scenario 2
The new scenario illustrates the non termination state (figure 5). We will redefine the body of the trigger T2 incorporating in its action a delete operation from TAB_PROF. As is shown below:

```
CREATE TRIGGER T2
AFTER DELETE ON TAB_TECH
WHEN C2
BEGIN ATOMIC
DELETE FROM TAB_PROF WHERE ……..;
END;
```

Now, if we examine this scenario, the operations sequence is similar to the previous scenario until the execution reaches the message (4: T2). In this time, the new statement incorporated into T2 is carried out after its execution. This operation (5: DELETE) generates the referential action execution (6:DC_TAB_PROF). As consequence, the last trigger operation (7: T1) is fired again. When a trigger is fired twice in the same scenario this means that the non termination state is detected. Therefore, the scenario is stopped and a message is sent to the developer which must resolve the problem and repeat the scenario again. The execution sequence of this scenario is shown below:

1: When DELETE FROM TAB_PROF is carried out ⇒

2: The DC_TAB_PROF is executed (section 2.3) ⇒

3: T1 (X=1) is executed ⇒

4: T2 is executed ⇒

5: DELETE FROM TAB_PROF is executed ⇒

6: Again 2 ⇒

7: Again 3 ⇒ ; STOP; *"Non termination state was detected. Please, solve the problem and try again"*

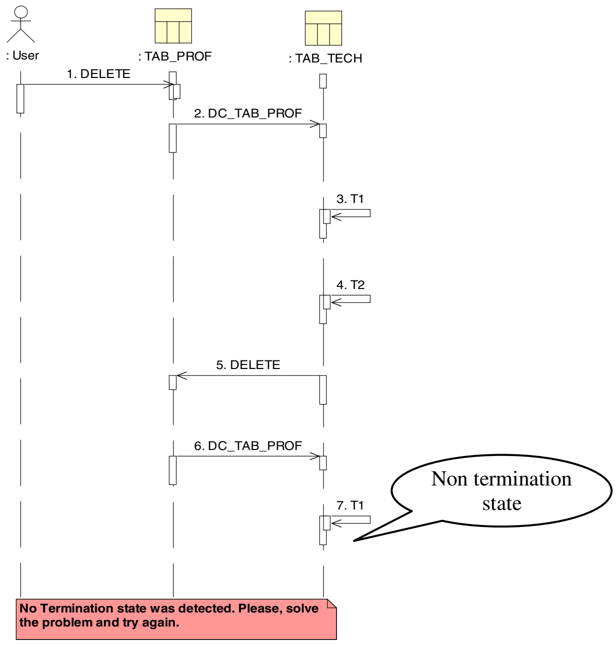

Fig. 3. Scenario 2, non termination state

5 Conclusions

Active rules/triggers systems are exposed in many studies and some challenges and issues are addressed to control the execution of these systems. One of these challenges is to encourage commercial CASE tools will cover all analysis phases with extended conceptual models.

Using triggers means additional effort in database development because the triggers execution model adds more complexity. We use UML's sequence diagrams to represent the triggers execution flow in order to verify the triggers behaviour and to guarantee the correct execution in Rational Rose Tool. Our principal objective in this work is to motivate database designers to use triggers for completing semantic specifications gathered in a conceptual schema through a CASE tool which shows triggers execution associated to a relational schema in an intuitive way.

As future work, we will extend our approach to include the confluence problem of triggers execution, and we will aggregate this tool into our toolbox in order to get to transform integrity constraints of a given schema into triggers. Furthermore, we are going to design some experiments to validate our tool and the efficiency use according to our contributions: to make easy the complex problem of triggers implementation and to check triggers execution.

References

1. Paton W. N., “Active Rules in Database Systems”, Springer-Verlag, New York, 1998.
2. Norman W. P., Diaz O., “Active Database Systems”, ACM Computing Surveys, Vol.31, No.1, 1999.
3. Ceri S., Cochrane R. J., Widom J., “Practical Applications of Triggers and Constraints: Successes and Lingering Issues”. Proc. of the 26th VLDB Conf., Cairo, Egypt, 2000.
4. A. Eisenberg, J. Melton, K. Kulkarni, J. Michels, F. Zemke, “SQL:2003 has been published”**,** ACM SIGMOD Record, Volume 33 , Issue 1, March 2004.
5. Melton J., Simon A. R.. “SQL: 1999 Understanding Relational Language Components", Morgan Kaufmann Publishers, 2002.
6. Hanson E. N., "The Design and Implementations of Ariel Active Database Rule System", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No.1, February 1996.
7. Dayal U., Buchmann A. P., Chakravarthy S., “The HiPAC Project” in Active database systems: triggers and rules for advanced database processing, Widom J., Ceri S., Eds. San Francisco, CA.: Morgan Kaufmann Publishers, 1996, pp. 177-205.
8. Widom J., Cochrane R. J., Lindsay B. G. "Implementing set-oriented production rules as an extension to Starburst". Proc. 7th International Conference on VLDB, September 1991.
9. Kulkarni K., Mattos N., Cochrane R., "Active Database Features in SQL3", Active Rules in Database Systems", Springer-Verlag, New York, 1998. pp 197-218.
10. Ceri S., Fraternalli, P., ”Designing database applications with objects and rules:the IDEA Methodology”. Addsion-Wesley”.1997.
11. E. Benazet, H. Guehl, and M. Bouzeghoub. “VITAL a visual tool for analysis of rules behaviour in active databases”, Proc of the 2nd Int. Workshop on Rules in Database Systems. Pages 182-196, Greece 1995.
12. 12. G. Kappel, G. Kramler, W. Retschitzegger. “TriGS Debugger A Tool for Debugging Active Database Behavior”, Proceedings of the 12th International Conference on Database and Expert Systems Applications, Springer-Verlag London, UK , 2001
13. O. Díaz, A. Jaime, N. Paton. “DEAR a DEbugger for Active Rules in an object-oriented context”. In M. Williams, N. Paton. Rules In Database Systems. Pages 180-193, LNCS Springer Verlag 1993.
14. Rational Web site http://www.rational.com/support/documentation/
15. AllFusion® ERwin® Data Modeler site http://www3.ca.com/Solutions/
16. ORACLE Web site http://www.oracle.com/technology/products/
17. Alexander A., Jennifer W., “Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism”, Proc. ACM-SIGMOD Conf. 1992.

18. Paton N., Díaz O., "Active Database Systems", ACM Computing Surveys, Vol.31, No.1, 1999.
19. Hickey T., "Constraint-Based Termination Analysis for Cyclic Active Database Rules". Proc. DOOD'2000: 6th. International Conference on Rules and Objects in Databases, Springer LNAI vol. 1861, July 2000, pp. 1121-1136.
20. Vadaparty, Kumar, "ODBMS - Bridging the Gap Between Objects and Tables: Object and Data Models", volume 12 - issue 2, 1999.
21. Timo Salo, Justin hill, "Mapping Objects to Relational Databases", Journal of Object Oriented Programming, volume 13 - issue 1, 2000.
22. UML 2 Sequence Diagram Overview http://www.agilemodeling.com/artifacts/sequenceDiagram.htm