PLUGGING ACTIVE MECHANISMS TO CONTROL DYNAMIC ASPECTS DERIVED FROM THE MULTIPLICITY CONSTRAINT IN UML

Harith T. Al-Jumaily, Dolores Cuadra, Paloma Martínez

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

Abstract. Multiple efforts have been devoted to face the problems of database modelling. One of them is the automatization of database design process using CASE tools. Frequently, these tools do not completely support all phases of database analysis and the design methodology that they propose. Therefore, we propose to incorporate new features to these tools enhancing them and solving some of the modelling problems. In this paper, we present an add-in module that aims to generate triggers for preserving the multiplicity constraints of a conceptual scheme in the transformation to a relational scheme. The module is integrated into the RATIONAL ROSE case tool.

1 Introduction

In database design methodologies such as [1] and [2], there are processes devoted to transform conceptual into logical schemata. In such processes, semantic losses are produced because logical elements are not coincident with conceptual elements. A correct transformation of constraints is necessary to preserve the semantics that reflects the Universe of Discourse. The multiplicity constraint, also called cardinality constraint [6], is one of these constraints that can be established in a conceptual schema. It has dynamic aspects that are transformed into the logical model as certain conditions to verify the insertion, deletion, and update operations. The verification of these constraints is a serious and complex problem because currently database systems are not able to preserve the multiplicity constraints of their objects.

The mechanism used to control this problem is the triggers/rules system. Because of the execution model of triggers is more complicated than traditional system we believe that automatic support to generate triggers is a good solution and can help the database developers. In this work, we present a module to generate triggers for multiplicity constraints verification that is integrated into RATIONAL ROSE case tool. This module can automatically create a SQL trigger-based script for any target DBMS; in this work, we consider only ORACLE DBMS.

In section (2) some active technology features are defined. Section (3) is devoted to explain the UML multiplicity constraints and some cases of semantic losses. In section (4) we discuss how the integration of the active technology into UML schema

is performed. In section (5), the add-in interface design is explained and finally some conclusions are exposed.

2 ACTIVE MECHANISMS (TRIGGER-BASED SYSTEM)

Although triggers are available in most DBMS, unfortunately the execution models of these triggers change from one DBMS to another. There are a number of common components that are valid for all systems [7] and these components usually are not changed. We explain the basic concepts according to the SQL 2003 standard which makes revisions to all parts of SQL99 and adds new features [8]. A trigger in the SQL standard 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. The standard SQL syntax for the creation of a trigger is shown in [9].
An event is a DML statement (INSERT/UPDATE/UPDATE) being issued against the associated table. An activation time (BEFORE and AFTER) defines whether the trigger is activated before or after the triggering event. Trigger granularity determines how many times the trigger is activated. There are two levels of granularity; a statement-level trigger that executes once for each triggering event and a row-level trigger that executes for each row that belongs to the modified set. If the modified set is empty then a row-level trigger does not execute, while a statement-level trigger executes once. Referencing values are used to create a new alias to reference the old and new values in a row being modified by triggering event. These values are accessible to the condition and the action of the trigger. As many previous works [7] [18], we used the triggering graph for detecting the non-termination state, and solving this problem by disabling the trigger that is again activated in the same activation set.

3 UML multiplicity

Since its introduction by Chen [5], the cardinality constraint is the number of entity instances that are associated in a relationship. Cardinality constraints are represented by minimum and maximum bounds [3]. UML multiplicity follows the Chen's style because to verify the cardinality constraints of one class we have to fix an object of the other class and to obtain how many objects are related to it [6]. Figure 1 shows an example of multiplicity constraints over a binary association using UML notation [12]. There are two associated classes (*A* and *B*) in the association *R*, if the association type is one-to-one then each class becomes a table, and the foreign key of *A* uses the primary key of *B*. Also, if the association type is one-to-many then each class becomes a table, and the foreign key of *A* uses the primary key of *B*. But if the association type is many-to-many then we need a new table *R* with primary keys of both classes *A* and *B*. Nulls are allowed in optional multiplicity but are not allowed in mandatory multiplicity.

In order to express the semantics of this scheme, rules are used in the logical scheme. These rules are the definition of primary key, foreign key, and cascade delete and update options. But, these options are not enough to control the minimum multiplicity constraint. Therefore, we must take care that the database is not in an inconsistent state every time that a DML statement modifies the database.

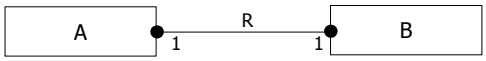

Fig. 1. Multiplicity Constraints using UML.

This task needs a big effort because the developers have to generate a big size of programs and to solve the complex execution model of triggers [4]. Also, in database design CASE tools such as Designer2000, ERwin, Rational Rose, the triggers generation is supported only to enforce referential integrity rules. The verification of the multiplicity constraints is not supported due to the complexity associated with the referential integrity rules. For example, Designer2000 does not generate any trigger for cardinality constraint validation, but it provides an editor (Trigger Definition), which allows defining different triggers by the users. ERwin and Rational Rose are able to create triggers to enforce the previous referential integrity rules, but not the multiplicity constraints. Also, they have an editor of triggers that allows defining triggers.

4 Description of the approach

In this section, we discuss how the integration of the active technology into UML scheme was done. In order to make easier to the reader the understanding of this approach, pseudocode is used to show the execution mechanisms and how the verification is performed. Our add-in module is integrated into Rational Rose CASE tool. The figure 2-Left shows an example of a simple database scheme using the UML class diagram elements. The mapping of each class and each association into UML Rational Rose Data Model (RRDM) (figure 2-Right) [15][16] and the integration of active technology are given below.

4.1 One-to-many associations

As shown in figure 2, there is a one-to-many association between the two persistent classes, Student and Department. This association defines that every student object must to have an association to one department object, and a department object requires an association to one or more student objects. The mapping to RRDM uses two tables, *tab_student, tab_department,* and a non-identifying relationship. The foreign key of *tab_student (pk_dpt)* uses the primary key of *tab_department* to build the relationship. In this work, we consider that only the minimum multiplicity of the class Department is mandatory. So, this foreign key has to be unique and not null. Therefore, when an event is generated to delete or update the foreign key values in *tab_student* we have to check that each department has an association with one or

more students. The following trigger shows the mechanism used to verify this constraint.

Create trigger trig_student_1
After delete or update(pk_dpt) on tab_student For each row
Begin
If $e_{student}$*=true then Select count(*) from tab_student where pk_ dpt = :old.pk_dp;*
If count(*)<$Mmin_{departmen}$ *then raise_error('cannot delete or update');*
End if;
End;

According to the execution model of triggers (section 2.), there are two events that activate a trigger associated to a child table which has the foreign key:

(1) $e_{department}$*=[delete or update(pk_dpt) on tab_department .cascade. delete or update(pk_dpt) on tab_student]*

When the event $e_{department}$ is generated, the trigger *trig_student_1* is consequently activated to the referential integrity rules (On Delete Cascade or On Update Cascade). In this case, it is not necessary to verify the semantics of the modified department because we are modifying the parent table *tab_department* and all related students in the child table *tab_student*. Therefore, whenever $e_{department}$ is generated, the trigger *trig_student_1* does not to verify anything.

(2) $e_{student}$*=[delete or update(pk_dpt) on tab_student]*

When the event $e_{student}$ is generated, the trigger *trig_student_1* is also activated. But, in this case the verification is more important because we need to assure that each department still has related students after the modification is performed. The related students to be in *tab_student* must have a count greater than the minimum multiplicity of Department $Mmin_{department}$. If the count is less than $Mmin_{department}$ then the trigger action rolls back the transaction and the database is restored to the state before the modification.

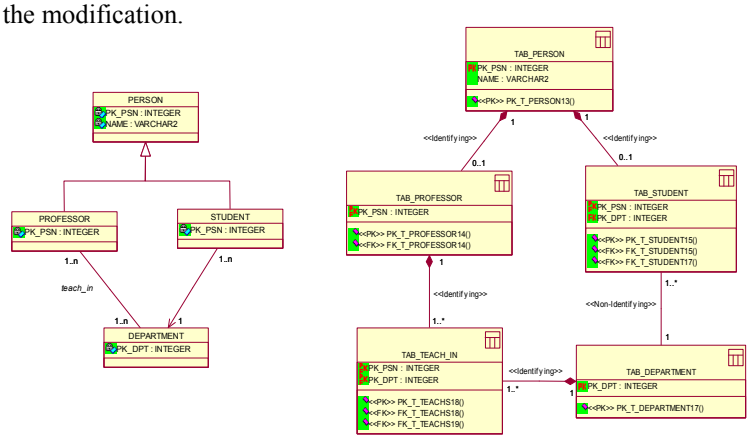

Fig. 2. Mapping UML objects model to RRDM.

In this work, we do not need to use triggers to verify the insertion because the maximum multiplicities of the classes that participate in the associations are unlimited (*n*). But, when the maximum multiplicity is known, the verification of the multiplicity is needed ever time that an insertion is done.

4.2 Many-to-many associations

Figure 2 shows that *teach_in* is a many-to-many association that relates the two persistent classes, Professor and Department. This association defines that every professor object has to teach in one or more departments, and a department object must to have an association to one or more professors. The mapping of this association into the RRDM uses two tables *tab_professor*, *tab_department*, and an identifying relationship that is mapped to a third table *tab_teach_in*. The foreign keys of *tab_teach_in (pk_prf, pk_dpt)* use the primary keys of the previous two tables to build the relationship. These foreign keys are not null, and the delete and update options can be cascade.

In this relationship to avoid the semantic loss we must check three constraints:

(1) $e_{department}$*=[delete or update(pk_dpt) on tab_department .cascade. delete or update(pk_dpt) on tab_teach-in]*

(2) $e_{professor}$*=[delete or update(pk_prf) on tab_professor .cascade. delete or update(pk_std) on tab_teach-in]*

(3) e_{teach_in} *=[delete or update(pk_prf, pk_dpt) on tab_teach_in]*

The following trigger is used to verify these constraints:

Create trigger trig_teach_in
After delete or update(pk_prf, pk_dpt) on tab_teach -in For each row
Begin
If $e_{department}$ *= false then Select count(*) from tab_teach-in where pk_prf = :old.pk_prf;*
If count(*)<$Mmin_{department}$ *then raise_error('cannot delete or update');*
End if;
If $e_{professor}$ *= false then Select count(*) from tab_teach-in where pk_dpt = :old.pk_dpt;*
If count(*)<$Mmin_{professor}$ *then raise_error('cannot delete or update');*
End if;
End;

In order to reduce the trigger body size and minimize the execution time, the verification of the event e_{teach_in} is included with the verification of the others events.

4.3 Generalization

The generalization is another type of relationship that has dynamic aspects to be verified. We consider the dynamic aspects of the Disjoint-Total constraints. Our example has one generalization that contains three persistent classes, Person, Professor, and Student. A person can be only a student or a professor. The approach that we use is to create one table per class. The *tab_person* contains all common attributes and a not null attribute (*is_a*) is used for partitioning. The identification (*pk_psn*) is used as primary key for the three tables. Also, it is assigned to be foreign key in *tab_professor*and *tab_student*. The referential integrity rules in this relationship are maintained by setting not null values for foreign keys and delete and update options as cascade option.

In this case, we preserve the semantics deleting an object from *tab_person* when *tab_professor* or *tab_student* is deleted. The following trigger is used to verify these constraints:

Create or replace trigger trig_student_2 After delete on tab_student
Begin
If $e_{student}$ *= true then Disable(trig_student_2);*
Delete from tab_person where pk_psn =:old.pk_psn;
End if;
End;
Note: The trigger trig_professor is similar to the previous trigger.

The activation of this trigger produces the non-termination problem because the trigger body contains an action that deletes from *tab_person*. When this action is executed the trigger is again activated and so on. In order to solve this problem, we have to disable *trig_student_2* if it is again activated in the same set.

5 Add-in Module Design

We applied our approach on the Rational Rose CASE tool because it is able to easily add-in software tools to support the development needs. Add-Ins can install menus, help files, contents tab file, properties, executables, script files, and OLE servers [13]. Our add-in module was developed using Basic Script Language [14], and can be accessed from the Tools menu. As shown in the figure 3, the add-in module has an interface that shows some options that we need to consider before generating triggers.

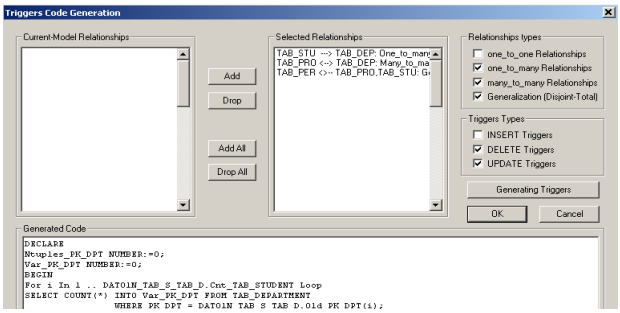

Fig. 3. Add-in Interface Design.

The add-in module detects and presents all relationships that belong to the current scheme. The list box "Current-Model Relationships" presents all relationships that exist in the current scheme. According to the required semantics, the user can choose those relationships that he needs to preserve. The list box "Selected Relationships" shows the selected relationships to be controlled. The user can choose one or more relationships to be controlled; as is shown in the figure we close all the relationships that belong to our example. According to our example, the add-in interface represents three type of relationships, one-to-many relationship (TAB_STU-->TAB_DEP) that associates *tab_student* with *tab_department*, many-to-many relationship (TAB_PRO<-->TAB_DEP) that associates *tab_professor* with *tab_department*, and generalization relationships (TAB_PER<>--TAB_PRO, TAB_STU) that associates the superclass *tab_person* with *tab_student* and *tab_professor*. The check boxes

"Relationship Types" show the relationship types that the add-in module considers. The check boxes "Triggers Types" represent the types of triggers to be generated. The generated code that we obtain is saved in a SQL file that contains triggers and packages to define the global variables.

Because of ORACLE triggers system has not Old/New-Table referencing values, and due to mutating table problem, we used two triggers for controlling each event. Therefore, each trigger shown in the section 4 is transformed into two ORACLE triggers. The first (Before/Row) is used to save the identification keys, and the second (After/Statement) is used to verify the semantics.

The add-in module plugs these triggers into an UML schema as operations using:

Set theOperation = theClass.AddOperation (OperationName, OperationType)

Figure 4 shows the plugged operations that have been generated to control the deleted events. Because of *tab_student* is associated to two relationships, it has four triggers. Two of them (*td1n_br_tab_student, td1n_as_tab_student*) are used for controlling the deletion of foreign keys that come from one-to-many relationship. The triggers *tdhy_br_tab_student, tdhy_as_tab_student* are used for controlling the constraints of the generalization relationship. The user could display and modify these triggers by browsing the table specification dialog box, as is shown in figure 5.

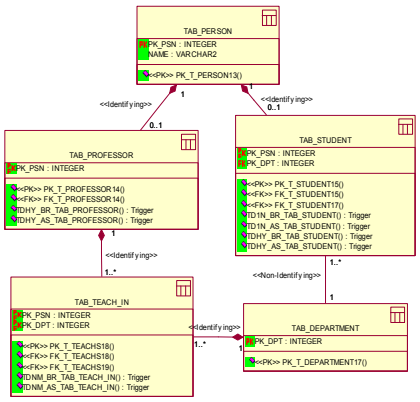

Fig. 4. Plugged operations into the schema

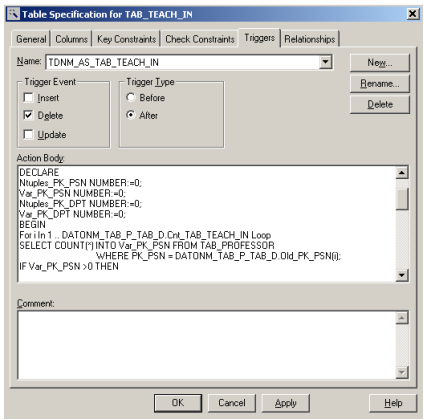

Fig. 5. Table specification dialog box

6 Some preliminary conclusions

The main contribution of our work is to provide a prototype as an add-in module for controlling dynamics aspects derived from the multiplicity constraint in Rational Rose. The verification of these constraints is very difficult. Therefore, we believe that incorporating add-in modules is a good solution to solve some of the modelling problems, and to enhance the tasks performed by database designers.

Currently, we are testing the performance of this approach in a real database and in the future, we will apply our work to preserve the multiplicity constraints in others DBMS adding their properties.

References

1. Elmasri, R. Navathe, S.: Fundamentals of Database Systems, Third Edition, Addison-Wesley, 2000.
2. Toby J. Teorey: Database Modeling & Design, third edition, Morgan Kaufmann Series in data management systems, 1999.
3. Teorey, T., Yang, D., Fry, J. A Logical Design Methodology for Relation Databases Using the Extended Entity-Relationship Model. Computer Surveys, Vol. 18. No. 2. 1986.
4. H. T. Al-Jumaily, D. Cuadra, P. Martínez, Applying a fuzzy approach to relaxing cardinality constraints. 15th International Conference on Database and Expert Systems Applications DEXA'04, Zaragoza, Spain, 2004.
5. Chen, P.: The Entity-Relationship Model – Toward a Unified View of Data, ACM Transactions on Database Systems, Vol. 1, N. 1. 1976.
6. D. Cuadra, C. Nieto, E. Castro, P. Martínez M. Velasco: Preserving relationship cardinality constraints in relational schemata, Database Integrity: Challenges and Solutions, Ed: Idea Group Publishing, 2002.
7. Norman W. P., Díez O., Active Database Systems, ACM Computing Surveys, Vol.31, No.1, 1999.
8. A. Eisenberg, J. Melton, K. Kulkarni, J. Michels, F. Zemke, SQL:2003 has been published, ACM SIGMOD Record, Volume 33 , Issue 1 (March 2004).
9. Jim Melton, Alan R. Simon, "SQL: 1999 Understanding Relational Language Components", Morgan Kaufmann Publishers, 2002.
10. Stefano Ceri, Roberta J. Cochrane, Jennifer Widom. "Practical Applications of Triggers and Constraints: Successes and Lingering Issues". Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.
11. Oracle9*i* Application Developer's Guide – Fundamentals, http://otn.oracle.com/documentation/index html
12. Unified Modelling Language Specification, Version 1-3, Object Management Group.
13. Rational Web site http://www rational.com/support/documentation/
14. BasicScript 2.25 Language Reference, Summit Software. http://www.cardiff.com/CSdownload/misc/t1057.pdf.
15. Timo Salo, Justin hill, Mapping Objects to Relational Databases, Journal of Object Oriented Programming, Issue: 2000 - volume 13 - issue 1.
16. Vadaparty, Kumar, ODBMS - Bridging the Gap Between Objects and Tables: Object and Data Models, Issue: 1999 - volume 12 - issue 2.
17. Gonzalo Génova, Juan Llorens, Paloma Martínez. The meaning of multiplicity of n-ary associations in UML. Software and Systems Modeling, vol 1, number 2, December 2002.
18. Ceri S., Fraternalli, P., Designing database applications with objects and rules : the IDEA Methodology, Addsion-Wesley,1997.