A New Tool to Check the Coherence of Constraints Defined on UML Class Diagrams

Djamel Berrabah[1], Faouzi Boufarès[2], Charles-François Ducateau[1]
[1]{berrabad,ducateau}@univ-paris5.fr

Centre de recherche en Informatique de Paris 5
Université Paris 5, 45 rue des Saints Pères
75270 Paris Cedex 06, France

Laboratoire d'Informatique de Paris-Nord
Université Paris-Nord, 99 avenue Jean-Baptiste Clément
93430 Villetaneuse, France

Summary
The unified modeling language (UML) is an important and efficient industry standard language for modeling databases. It is supported by most of CASE tools available on the market. However, these tools do not preserve the efficiency of UML during translation. In other words, they often do not take into account all the information (structures and constraints) given in a UML class diagram. The goal of this paper is to enrich and improve these tools in order to solve this problem. So we aim to propose an efficient approach to generate automatically mechanisms that check participation constraints. These mechanisms are active during the maintenance of databases. If any operation brings about an inconsistent database state (i.e. violates constraints) it will be rejected and the data of the database will remain unchanged

Key words:
UML for databases design, class diagram translation, active database, integrity constraints checking.

1. Introduction

The Unified Modeling Language (UML) [20, 22] became a powerful and a commonly used formalism for database (DB) analysis and design. Most of CASE tools available on the market, such as Power AMC, Rational Rose and DB-Main, [10, 24, 26], are improved to support UML. They produce a flexible environment for data modeling. Despite these features, designer's needs are not totally satisfied. For instance, some constraints, such as disjointness, covering and Exclusion, are neither expressed nor translated by any CASE tool. CASE tools are based on database design methodologies [13, 27] to translate a UML class diagram into a relational schema (RS). Class diagrams are also called conceptual schemas (CS). The RS elements obtained during translation processes do not completely coincide with the CS elements, thus bringing about some semantic losses [4]. This problem often arises when most of the constraints that are established in the CS and that reflect the real world are not translated correctly. The standardization of SQL is still work in progress. However, the latest revision of SQL [11] did not bring significant improvement with regards to participation constraints, and many other constraints.

Our aim through this paper is to study participation constraints which are used in a class diagram. These constraints can be defined on binary relationships as well as on generalization/specialization relationships. Their behaviors are not the same in either case, even though they have the same semantics. Our contribution in this work is to give a means to express and translate automatically these constraints using event-condition-action (ECA) rules. Of course, the implementation of well-known constraints in relational databases using ECA rules or triggers has already been thoroughly studied. The main goal of our study is to develop a new CASE tool (Fig.1). This tool provides the possibility to take all constraints into account (their global coherence and their translation) within database analysis and design process.

This paper is organized as follows. Section 2 synthesizes the basic principles of constraints and their role in preserving the semantics of the real world. Section 3 introduces participation constraints in UML. Sections 4 and 5 show how to express and translate participation constraints on binary relationships and generalization/ specialization relationships with a constraint specification language. Our study is based on trigger-based SQL scripts. Section 6 contains our conclusions and prospects.

2. Integrity Constraints Checking

2.1 Constraints

Maintaining the database consistency is a very interesting challenge. Generally, the consistency is enforced by

Manuscript received August 25, 2006.
Manuscript revised August 30 , 2006.

integrity constraints (ICs), which are assertions that database instances are compelled to obey. A database is consistent if and only if all ICs are satisfied. ICs have been classified according to various criteria. The first criterion distinguishes between state constraints, which characterize valid database states, and transition constraints, which impose restrictions on the possible transition state of a database. The problem of checking integrity constraints is not recent. Many works related to this problem can be found in the literature [6, 7, 9]. The authors in [12, 15] represent an attractive synthesis of different approaches related to IC checking. Indeed, two major levels can appear throughout database design. The first one is related to conceptual schema checking. On this level, the CS syntax and especially its semantics must be checked [4, 19]. For instance, an XOR constraint defined on two relationships can be in contradiction with multiplicity constraints; it therefore remains unchecked. On the second level, the CS is translated in a target language. This translation can be made into a formal specification such as Z and B [17, 18, 23] or in a query language such as SQL [1, 5]. These works concern data structures (classes, relationships, etc…) and chiefly multiplicities in constraints. Few works deal with the participation constraints. Active mechanisms are used in [2] to translate the participation constraints defined on binary relationships. This same kind of constraints can be defined on the relations of heritage.

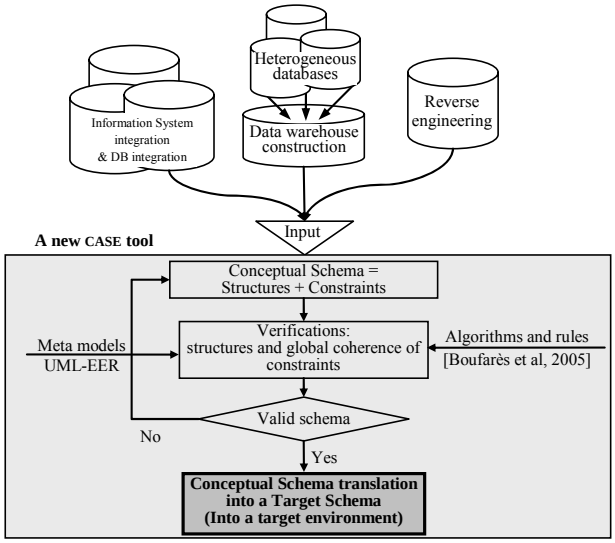


Fig. 1 A new tool to integrity constraint checking

2.2 Triggers

Some conditions are checked by declarative constraints. Sometimes, however, declarative constraints are not sufficient. Thus more powerful systems such as triggers [9] have to be used. Triggers constitute a good means to implement referential actions. On the majority of DBMSs, it is necessary to use triggers to perform actions other than those defined by defect. Though triggers are found in the majority of DBMSs, unfortunately the execution models of the triggers vary from one DBMS to another. Main components are valid for all the systems and generally do not change. In SQL 2003 [11], a trigger is expressed by ECA rules [8, 9, 16]. It is activated during DB transition state. Each trigger is associated to one or more events on a table. It is activated if one of these events is performed on this table. An event can be an IDU (Insert, Delete or Update) statement applied on a table of the DB. Once the trigger is activated, its condition, that is to say an assertion on the data or the state of the DB, must be evaluated. If the condition is evaluated as “true”, then the action is performed. An action is a sequence of SQL statements performed on the DB tables or a "raise error" which rejects the event that activated the trigger. If the event is rejected, the data of the DB do not change. Triggers can access the old and new attribute values affected by the triggering event by means of transition variables (OLD and NEW) [6].They are noted by X.attribute_name, with X in {OLD, NEW}, and attribute_name is a table column.

3. Participation constraints in UML

A participation constraint (PC) frequently relates to the coexistence of class object occurrences in one or several associations (Fig.2). Several participation constraints are

presented in the literature, such as exclusion, inclusion, etc... More detailed definitions of these constraint categories are given by [3, 21]. These constraints, once introduced into a CS, must be taken into account in order to preserve the semantics of the real world. This must be done at the conceptual level as well as at the relational one.

Two kinds of participation constraints are studied In this paper: those defined on classical binary relationships and those defined on generalization/ specialization relationships. In other words, rules are provided to allow designers to define and to express these constraints using a constraint specification language. Participation constraints basically refer to conditions of linking class objects to two or several other class objects. The first aim of this study is to remove ambiguities on definitions of these constraints when defined on generalization/ specialization or binary relationships. The second one is to underline their common points. Finally, we show how to express them (in trigger script SQL) and check them.

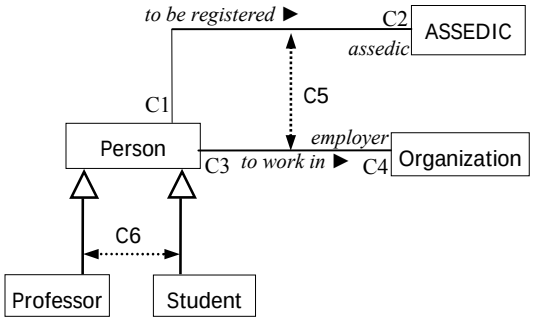


Fig. 2 Human resources management Schema

The set of constraints: ζ
C1: 1..*
C2: 0..1
C3: 1..*
C4: 0..1
C5: exclusion
C6: disjoint, incomplete

The conceptual schema in figure 2 describes human resources management. Examples of both kinds of PCs are represented in this figure. C5 is a PC defined on a binary relationship. It is an exclusion constraint that ensures that each person must, at least, work in an organization or be registered in ASSEDIC[1] but not both at the same time. C6 is a PC defined on generalization/ specialization relationship. It is a disjoint and complete constraint which guarantees that a person may be either a professor or a student but not both. She/he may be neither. We have shown in [4] that the CS is valid if and only if the set is coherent. Otherwise, the translation of the conceptual schema to the relational one must not be done. For instance, if C2 is equal to "1" then the constraint exclusion has no signification.

[1] French term : "**ASS**ociation pour l'**E**mploi **D**ans l'**I**ndustrie et le **C**ommerce" means Organization for Employment in Industry and Trade

4. Checking PCs on binary relationships

4.1 From conceptual schema to relational one

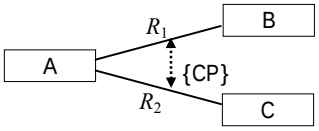


Fig. 3 Participation constraints defined on a binary association

Translating and checking the participation constraints strongly depend on the conceptual schema translation and consequently on multiplicity constraints. A multiplicity constraint relates to minimal and maximal numbers of links that can exist between the objects of the associated class. Many algorithms to map conceptual to relational schemas can be found in the literature [13]. Only rules considered to be useful for our study are presented in the following forms:

- Rule 1: Any class is transformed into a table with a primary key.
- Rule 2: Any binary association which does not contain a maximum multiplicity equal to 1 is represented by a table. Its primary key is composed by the primary keys of the concerned classes. These primary keys constitute foreign keys too.
- Rule 3: Any binary association with a maximum multiplicity equal to 1 is represented in the form of a foreign key.

Three classical types of multiplicity couples are considered (Fig.4): 1) one-to-many: Only one multiplicity has a maximum equal to 1 noted 1-N or N-1. 2) one-to-one: Both multiplicity constraints have a maximum equal to 1 noted 1-1. This case is similar to the 1-N one. 3) many-to-many: All maximum multiplicity constraints are not equal to 1 noted N-M.

Figure 4 summarizes the Relational sub-Schemata associated with the CS of figure 3 according to the couples of multiplicity constraints defined on its associations. In other words, this figure represents only the tables containing occurrences of associations on which PCs are defined. PKX means the Primary Key of the table X, FKX means the Foreign Key of the table X, AttrX means the

Attributes referencing the table X and AttrR means the Attributes of association R.

Case	Association Type	relational Sub-Schema
CaseI	N-1 & N-1	A(PKA,FKB,FKC, AttrA)
CaseII	N-M & N-1	T(FKB, FKA, AttrR) A(PKA, FKC, AttrA)
	1-N & N-1	T= B(PKB, FKA, AttrB) A(PKA, FKC, AttrA)
	N-1 & 1-N	A(PKA, FKB, AttrA }) T= C(PKC, FKA, AttrC)
	N-1 & N-M	A(PKA, FKB, AttrA) T(FKC, FKA, AttrR2)
CaseIII	N-M & N-M	T1(FKB, FKA, AttrR1) T2(FKC, FKA, AttrR2)
	1-N & N-M	T1= B(PKB, FKA, AttrB) T2(FKC, FKA, AttrR2)
	N-M & 1-N	T1(FKB, FKA, AttrR1) T2= C(PKC, FKA, AttrC)
	1-N & 1-N	T1= B(PKB, FKA, AttrB) T2= C(PKC, FKA, AttrC)

Fig. 4 Summary of the RS associated to CS according to the various multiplicity constraints

Three different cases can be distinguished in figure.4. Case.I) the objects of both associations appear in the table A. Case.II) the objects of only one association appear in the table A. The objects of the other association appear in the table T. Case.III) no objects of either association appear in the table A. So the objects of both associations appear respectively in T1 and T2. More detail is given in subsection 4.2.

4.2 Active mechanisms generation

This subsection describes how triggers are automatically generated to check PCs defined on binary relationships. These triggers are represented in ECA rule form.

Case I

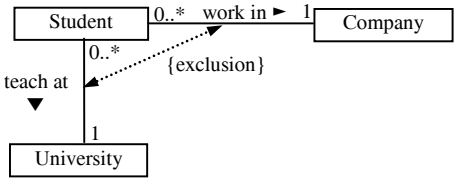
This case represents the N-1 & N-1 association types. Thus, associations R1 and R2 are both translated by the migration of the primary keys of classes B and C respectively as foreign keys (FKB and FKC) in the table A (Fig.4). With this solution, all the objects of both associations appear in the table A.

Exclusion constraint

In this case (Case.I), the exclusion constraint is violated only if an A-object participates in one association while it already participates in the other. This can occur during an insertion or an update operation. In order to solve the problem, it is necessary to generate a trigger that reacts to these events on the table A. The deletion operation has no effect on this constraint.

```
event: insert or update on A
condition: new value of FKB is not null and
new value of FKC is not null
action: raise error
```

Example 1



In this example, a student either teaches at the university or works in a company but not both at the same time. He may not do either. To ensure this condition it is necessary to add an exclusion constraint between the associations "work in" and "teach at". "work in" and "teach at" are both many-to-one associations (N-1 & N-1). The trigger generated in this case is as follows:

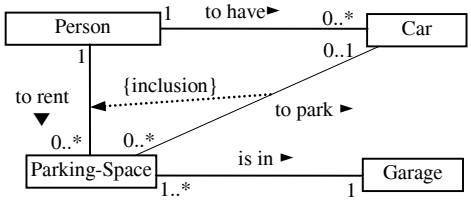
```
Create trigger Insert_Update_Student
Before insert or update on Student
Begin
   If (NEW.FK_COMPANY IS NOT NULL AND
       NEW.FK_UNIVERSITY IS NOT NULL)
   Then RAISE_ERROR ('exclusion constraint
       violated');
   End If;
End Insert_Update_ Student;
```

Inclusion constraint

In Case.I, only the insertion and the update operations can violate the inclusion constraint. Only one trigger needs to be generated in order to prevent these events from violating this constraint. The trigger rejects the event if the new value of FKB is different from the null value and if that of FKC is null.

```
event: insert or update on A
condition: new value of FKB is not null and
new value of FKC is null
action: raise error
```

Example 2



In this example, a parking-space is rented by a person to park a car. The car cannot be parked if the parking-space is not rented. To ensure this condition it is necessary to add an inclusion constraint between the associations "to rent" and "to park". This constraint is expressed by the following trigger:

```
Create trigger Insert_Update_Parking-Space
Before insert on Parking-Space
Begin
If NEW.FK_CAR IS NOT NULL AND FK_PERSON IS
   NULL)
Then RAISE_ERROR ('inclusion constraint
            violated');
End If;
End Insert_Update_Parking-Space;
```

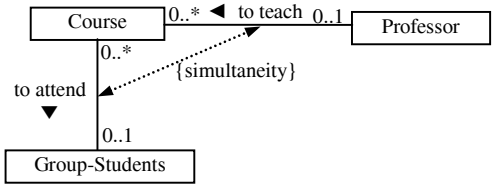
Simultaneity constraint

In Case.I, two solutions are possible for this constraint. The first solution consists in saving each IDU statement performed on the tables where the R1 and R2 objects appear. Thus a new table will be created in which these operations are saved. This new table will be dealt within a procedure executed by the user in order to check whether the simultaneity constraint is satisfied or not. Some of the disadvantages of this solution are that it is semi automatic and storage space consuming. The second solution somehow lacks flexibility. It generates IDU statements that the user must perform, following other IDU statements (for example, if the user inserts an A-object in one of the two tables, then he must also insert it in the other one).

In Case.I, deletion operation has no effect on this constraint. The two events which can violate it are insertion and update. To perform one of these two operations, the values of the foreign keys in the table A must both be (at the same time) either null or not null. The trigger that must be generated to check this condition is as follows:

event: insert or update on A
condition: new value of FKB is different then new value of FKC and at least one of them is null
action: raise error

Example 3



It is considered in this example that a course is taught by at most one professor and is attended by at most one group of students. A course may be neither taught nor attended. On the other hand, if it is attended by a group of students it must be taught by a professor and vice versa. To ensure this condition, a simultaneity constraint must be added between associations "to teach" and "to attend".

```
Create trigger Insert_Update_Course
Before insert or update on Course
Begin
  If (NEW.FK_PRO IS NULL AND NEW.FK_GROUP IS
     NOT NULL) OR (NEW.FK_PROFESSOR IS NOT
     NULL AND NEW.FK_GROUP IS NULL)
  Then RAISE_ERROR ('simultaneity constraint
     violated');
  End If;
End Insert_Update_Course;
```

Totality constraint

In Case.I, checking this constraint is quite easy. If an insertion or update operation is performed, it is enough to check that at least one of the two values of the foreign keys in A is not null. If both are null, this means that the A-object does not participate in any association and consequently the constraint is violated. The operation of deletion does not violate this constraint, because if it is performed on the table A, it eliminates some A-objects as well as their participations.

event: insert or update on A
condition: new value of FKB is null and new value of FKC is null too
action: raise error

Case.II

In this case, one of the two associations is translated by a foreign key (FKB or FKC) in the table A (Fig.4). The A-object participation in one of the two associations appears in table A. The tables taken into account in this case are classified in Figure.4 Case.II.

Exclusion constraint

In Case.II, three events can violate the exclusion constraint. These events are an update on table A, an insertion or an update on the table T. Two triggers must be generated to prevent the violation of this constraint. The first reacts to an update on table A. Its principle is to reject this update if the new value of the foreign key in the table A is different from the null value and the value of PKA already exists in the table T. This trigger is as follows:

event: update on A
condition: new value of FKB is not null and the set of rows that T.FKA= A.PKA is not empty
action: raise error

The second trigger reacts to an insertion or an update on the table T. Its principle is to reject these two events if the value of FK, with which the new value of FKA is associated, is different from the null value.

event: insert or update on T
condition: new value of FKA is not null the FK value is not null where A.PKA=T.FKA
action: raise error

Inclusion constraint

To deal with this constraint in Case.II, two different sub-cases must be considered. Knowing that inclusion constraint is defined in such way that:

R1⊂R2 (i.e. T⊂A (Case.II-1) or A⊂T (Case.II-2)).

The first sub-case is summarized in the N-M&N-1 and 1-N&N-1 association types. The second one is summarized in the N-1&1-N and N-1&N-M association types.

Sub Case.II-1

In this sub-case, a deletion operation does not violate the inclusion constraint. An insertion or an update operation violates this constraint only if it is applied on the table T. To prevent this violation, a trigger is generated that rejects the insertion or the update of an A-object in the table T if its participation does not appear in the table A. Its definition is as follows:

```
event: insert or update on T
condition: N.FKA value is not null and A.FKC
value is null where T.FKA=A.PKA
action: raise error
```

An update operation can also violate the inclusion constraint if it is applied on the table A. A trigger must be generated too to reject this operation if it cancels A-object participation from the table A while this object participates in table T. The trigger is as follows:

```
event: update on A
condition: value of FKA is not null and FKC
is null where T.FKA =A.PKA
action: raise error
```

Sub Case.II-2

In this sub-case, four events can violate the inclusion constraint. The first one is an insertion operation in the table A. The following trigger, generated to prevent this violation, rejects this operation if the value of FKB is not null, which means that the A-object participates in the table A and does not participate in the table T.

```
event: insert on A
condition: value of FKB is not null
action: raise error
```

The second event that can violate the inclusion constraint is a deletion operation on the table T. The trigger generated here rejects this operation if it cancels the participation of an A-object in the table T while this object participates in the table A.

```
event: delete on T
condition: Old value of FKA is unique in T
and A.FKB value is not null where
T.FKA=A.PKA
action: raise error
```

The third event is an update operation on the table A. The trigger generated here rejects this operation if it creates a participation of an A-object in the table A although this object does not participate in the table T (A⊄T).

```
event: update on A
condition: new value of FKB is not null and
T.FKA value is null where A.PKA =T.FKA
action: raise error
```

Finally, the last event that can violate this constraint is an update operation on the table T. In this case, the trigger generated rejects this operation if it cancels the participation of an A-object in the table T although this A-object participates in the table A.

```
event: update on T
condition: new value of FKA is null and
A.FKB value is not null where T.FKA= A.PKA
action: raise error
```

Simultaneity constraint

Preserving simultaneity constraint, in Case II, is complex because each operation has a specific effect on tables A and T. For an insertion operation in table A, if the value of FK is non-null, then it is necessary to insert a row in the table T with the value of PKA in the FKA column. Consequently, insertion in the table T does not violate the constraint.

```
event: insert on A
condition: new value of FK is not null
action: insert value of PKA in T.
```

Deletion in the table A does not present any risk of violation of the simultaneity constraint because if a row is removed from the table A, then all the referenced rows are removed too. On the other hand, deletion from the table T can violate this constraint. If the value of FKA, in the removed row, is not null and unique, then_simultaneity cannot be ensured.

```
event: delete on T
condition: old value of FKA is not null and
unique
action: raise error
```

An update operation on the two tables A and T has an effect on the simultaneity constraint. Thus, to update a row in the table A, it is necessary to deal with the old and new values of FK (the foreign key). The value of FK determines if A-objects participate or not in one of the two associations. Therefore, a trigger is needed to deal with the change of this value. It is as follows:

```
event: update on A
condition1: new value of FK is not null and
PKA value do not exist in T
action1: insert value of PKA in T.
condition2: old value of FK is not null and
unique and new value of FK is null
```

```
action2: delete lines that contain PKA
```

Updating the table T can modify the participation of an A-object in the association transformed in T. Thus, a trigger is needed to reject this operation if it generates the participation of an A-object in an association while this object participates in the other and vice versa.

```
event: update on T
condition1: new value of FKA is not null and
A.FK value is null where A.PKA=T.FKA
action1: raise error
condition2: old value of FKA is not null and
unique and different than new value of FKA
action2: raise error
```

Totality constraint

In Case.II, an insertion operation has an effect on this constraint only if it is applied on the table A. The totality participation of all A-objects, either in one association or in both associations at the same time, must be satisfied during their insertions. This is ensured by a trigger that inserts the A-objects in the table T if they do not appear in the table A. This trigger is identical to one generated on the insertion operation for the simultaneity constraint. If a row is removed from the table A, then the totality constraint is not violated, but if a row is removed from the table T, then the constraint may be violated. This occurs when the removed row represents the unique participation of an A-object, which violates the constraint. The trigger that prevents this violation is as follows:

```
event: delete on T
condition: old value of FKA is not null and
unique and value of A.FK is null where
A.PKA=T.FKA
action: raise error
```

An update operation on the table A can also violate the totality constraint because the participation of an A-object can appear in only one row in the table A, and the update operation can cancel this participation. Thus, a trigger becomes necessary to preserve this constraint.

```
event: update on A
condition: old value of FK is not null and
new value of FK is null and PKA value do not
exist in T
action: raise error
```

In the same way, for an update in the table T, it is necessary to make sure that the updated row does not represent a unique participation of an A-object. The generated trigger rejects this operation if this occurs.

```
event: update on T
condition: old value of FKA is not null and
new value of FKA is null and FK value is
null where A.PKA=T.FKA
action: raise error
```

Case.III

In this case, none of the association will be translated by a foreign key in the table A (i.e. the participations of A-objects will not appear in the table A). Let us consider two tables T1 and T2 which represent respectively the transformation of the associations R1 and R2. The tables taken into account in our study are classified, in the table above, according to the association types (Fig.4 Case.III).

Exclusion constraint

Four events can violate the exclusion constraint in Case.III, an insertion or update of the table T1 and an insertion or update of the table T2. Therefore, two triggers must be generated in order to control this constraint. These two triggers have the same principle. The one defined on the table T1 (resp. T2) rejects the events (Insert/Update) if the new value of FKA already exists in the table T2 (resp. T1).

```
event: insert or update on T1
condition: new value of FKA exist in T
action: raise error
```

Inclusion constraint

Four events can violate this constraint in Case.III; an insertion or update of the table T1 and a deletion or update of the table T2. Therefore two triggers should be generated. The first one deals with the new values of FKA in T1 and the second one deals with the old values of FKA in T2. They are defined as follow:

```
First trigger:
event: insert or update on T1
condition: new value of FKA do not exist in
T2
action: raise error

Second trigger:
event: delete or update on T2
condition: old value of FKA exist in T2
action: raise error
```

Simultaneity constraint

If an occurrence of an A-object appears in the table T1 it must also appear in the table T2 and conversely. Thus two triggers should be generated to ensure this constraint during the insertion, one on T1 and the other on T2. These triggers have the same principle.

```
event: insert on T1
condition: T1.FKA value do not exist in T2
action: insert T1.FKA value in T2.
```

A deletion operation can violate the simultaneity constraint if it is performed on the table T1 or on the table T2. Thus the trigger defined on T1 (resp. T2) checks if the row to be removed represents the single participation of an A-object, then it removes all rows in T2 (resp. T1) that contain the A-object.

```
event: delete on T1
condition: old value of T1.FKA is unique in
T1
action: delete from T2 all rows that contain
T1.FKA value OR raise error
```

Two triggers must be defined to control the update operation. To deal with the new value of FKA, they look like the trigger defined on the insertion operation shown above. To deal with the old value of FKA, they look like the trigger defined on the deletion operation also shown above.

```
event: update on T1
condition1: T1.FKA value do not exist in T2
action1: insert T1.FKA value in T2.
Condition2: old value of T1.FKA is unique in
T1
Action2: delete from T2 all rows containing
T1.FKA value OR raise error
```

Totality constraint

Preserving this constraint during the insertion of the A-objects is necessary. The trigger generated here proposes to the user to choose one of the two tables T1 or T2 in order to insert the A-object which he wants to insert in A.

```
event: insert on A
condition: true
action: insert in A.PKA in T1 or T2.
```

The deletion operation can violate the totality constraint. It is necessary to generate two triggers on the two tables T1 and T2. These two triggers have the same principle. The trigger, definite for example on T1, rejects the operation if the row to be removed is the unique row containing the participation of an A-object in both tables T1 and T2. During an update operation, it is enough to check the old value of FKA, which is equivalent to a deletion operation. Dealing with the new value is equivalent to an insertion operation, which does not violate the totality constraint.

```
event: delete or update on T1
condition: old value of T1.FKA is unique and
do not exist in T2
action: raise error
```

5. Checking PCs on generalization / specialization relationships

5.1 PCs on generalization / specialization relationships

The generalization/specialization relationships concept was invented by Smith [25]. It can arise from superclass/subclass hierarchies in semantic data modeling [13]. These data structures have been widely studied and applied in many fields of computer science. Their popularity has grown from programming language to databases and from analysis and design methods to user interfaces [14]. In the field of database, it is called data structure. Actually, it is straightforward to interpret this data structure. For instance, it is possible to assert that a person in figure.2 denotes a more general concept than student or professor, and conversely, student and professor represent specialized concepts with respect to person.

This data structure may be accompanied by constraints which can be defined implicitly as well as explicitly. In the first definition, every object can belong to the generalization as well as to its specialization. Each special object has a link within exactly one general object but the reverse does not necessarily hold. Thus, the special objects indirectly have features of the more general objects. The explicit constraints are called participation constraints. We aim, in this subsection, to show how to express and check these constraint categories. To facilitate this task, we represent a generalization/ specialization relationship as a one-to-one relationship strengthened with an additional constraint (Fig.5). This additional constraint must ensure that each object of the specialization is related to an object of the generalization and that an object of the generalization may be associated to at most one object in the specialization.

Our proposition is indeed not to translate the Generalization/ Specialization relationship (structure) since many solutions are proposed to do their translation. Our principal goal is to check all categories of constraints defined in this kind of relationships.

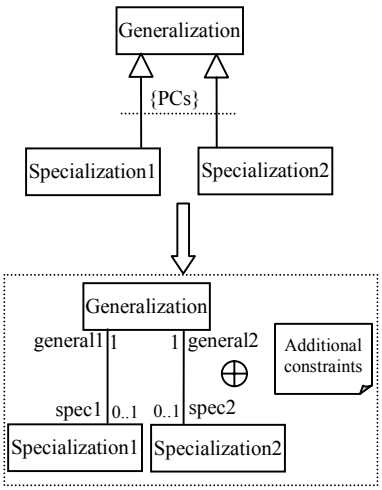


Fig 5 PCs on Generalization/Specialization

{PCs} is {disjoint, incomplete} or {overlapping, complete} or {disjoint, complete}

In the relational model, generalizations and specializations are translated into tables. The multiplicity "one" ensures that each special object is related to one and only one general object. Maximum multiplicity is translated by the migrating the primary key attributes of the generalization as foreign key into the specialization. Minimum multiplicity is translated into a not null constraint on that attribute. In addition, this solution allows to connect a general object to more special objects. This solution does not respect the definition of this data structure. Consequently, to guarantee that a general object is linked to at most one special object, we must define a unique constraint on the foreign key attribute in the specialization.

PCs defined on generalization / specialization relationships are divided into disjoint and complete constraints. A disjoint constraint specifies whether two objects of different specializations may be related to the same object of the generalization. A complete constraint specifies whether objects of the specifications are related to all generalization-objects (Fig.5). To check these constraints, we translate them automatically using triggers. These triggers are represented in ECA rule form.

5.2 A disjoint constraint

A PC is disjoint if each general object has a link to at most one special object. In the reverse case, this constraint is said to be overlapping. To check the disjoint constraint, we must ensure that each general object is a member of at most one specialization. This check must be done during the maintenance of data. To do this, a trigger is generated, whose events are an insertion or updating of an object in a specialization. The action of this rule is to reject the operation if that object already exists in the other specialization.

```
event: insert or update on Specialization1
condition: new value of object to be insert
or update exists in Specialization2
action: reject operation
```

5.3 A complete constraint

A PC defined on a generalization / specialization relationship is complete if each general object has a link to at least one special object, otherwise it is said to be incomplete. This can be ensured by adding three trigger which state that if a general object is not related to any object in one specialization it must be related to an object in the other. The first trigger checks the PC during insertion into the generalization. The second and the third ones check the PC during deletion or updating in specializations. They are identical.

```
First trigger
event: insert on Generalization
condition: none
action: insert obligatory object in
Specialization1 or Specialization2

Second trigger
event: delete or update on Specialization1
condition: old value of object to be delete
or update do not exists in Specialization2
action: reject operation
```

5.3 A disjoint and complete constraint

A PC defined on a generalization/specialization relationship may be disjoint and complete at the same time. In this case each general object must be related to at least one special object and at most to one object in both specializations. This case joins together the two preceding cases, and from there it is possible to use additional constraints to express disjoint and complete PCs. That assertion can also be expressed using triggers as follows:

```
First trigger
event: insert on Generalization
condition: none
action: insert obligatory object in one in
only Specialization

Second trigger
event: delete on Specialization1
condition: none
action: delete object from generalization
 or insert it in Specialization2

Third trigger
event: update on Specialization1
condition: none
action: delete new object from
Specialization2 if it exists there or reject
operation delete old object from
Generalization or insert it in
Specialization2
```

6. Conclusion and perspectives

In this paper, we reported a systematic study of the use of participation constraints for the specification of assertions defined on the behavior of class object participations. Sometimes, it is necessary to use these constraints in a conceptual schema to satisfy customer requirements. Our aim was to remove any ambiguity from the definition of participation constraints. These kinds of constraints, though defined on binary relationships as well as on generalization, have the same semantics but not the same behavior. We have translated the two categories of

participation constraints using trigger-based SQL additional. Thus, we have provided a general framework for transforming all kinds of participation constraints.

We are completing our data modeling prototype by integrating a great number of constraints. This prototype first checks the global coherence of constraints defined in the conceptual schema [4] and then translates all constraints in a specific language such as OCL, SQL or Others [2, 3].

References

[1] Al-Jumaily, H.T., Cuadra, D., Martinez, P. "Plugging Active Mechanisms to Control Dy-namic Aspects Derived from the Multiplicity Constraint in UML. The workshop of 7th International Conference on the Unified Modeling Language, Portugal (2004).

[2] Berrabah, D., Boufarès, F., Ducateau, C.F. Active Mechanism for Checking Participation Constraints in UML. International Conference on Enterprise Information Systems, Paphos, Cyprus (2006). (To appear)

[3] Berrabah, D., Boufarès, F., Ducateau, C.F. Analysing UML Graphic Constraint, How to cope with OCL. 3rd International Conference on Computer Science and its Applications, California USA (2005), p 74-82.

[4] Boufarès, F, Berrabah, D., Ducateau, C. F., Gargouri, F.: Les conflits entre les contraintes dans les schémas conceptuels de Bases de Données: UML – EER. Journal of Information Sciences for Decision Making, Special Issue of the 8th MCSEAI'04, N°19 (2005) Paper number 234.

[5] Boufarès, F. and Kraïm,N. Anew tool to analyse ER-schemas. Second Asia-Pacific Conference on Quality Software. Hong Kong China (2001). P 302-307.

[6] Ceri, S., Cochrane, R.J. and Widom, J. Practical Application of Triggers and Constraints: Successes and Lingering Issues. In Proc. of the 26th International Conference on Very Large Data Bases, Brisbane Australia (2000), p .254-262.

[7] Ceri, S., Fraternali, P., Paraboschi, S. and Tanca, L. Automatic Generation of Production Rules for Integrity maintenance. ACM Transactions on Database Systems, vol. 19, Issue 3, September 94.

[8] Ceri, S. and Widom, J.: Deriving production rules for constraint maintenance. In Proc. of the 16th International Conference on Very Large Data Bases, Brisbane Australia (1990), p 566-577.

[9] Cochrane, R.J., Pirahesh, H. and Mattos, N.M.: Integrating triggers and declarative con-straints in SQL database systems. In Proceedings of the 22nd International Conference on Very Large Data Bases, Mumbai India (1996), p 567-578.

[10] DB-Main, http://www.db-main.be

[11] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J., Zemke, F.: SQL: 2003 has been publis-hed. ACM SIGMOD Record, Volume 33, Issue 1, March (2004).

[12] Fahrner, C., Marx, T. and Philippi, S. DICE: Declarative Integrity Constraint Embedding into the Object Database Standard ODMG-93. Data & Knowledge Engineering, vol. 23, Issue 2, p 187-223. (1997).

[13] Elmasri, R., Navathe, S.: Fundamentals of Database Systems. 4th ed., Addison-Wesley (2003).

[14] Formica, A. and Missikoff, M. Inheritance processing and conflicts in structural generaliza-tion hierarchies. ACM Computing Surveys (2004), p 263-290.

[15] Grefen, P.W.P.J. and Apers, P.M.G. Integrity Control in Relational Database Systems – An Overview. Data & Knowledge Engineering, vol. 10, p 187-223. (1993).

[16] Horowitz, B.: Intermediate states as a source of non deterministic behavior in triggers. In 4th International Workshop on Research Issues in Data Engineering: Active Database Sys-tems, Houston TX February (1994), p 148-155.

[17] Laleau, A., Mammar, A.: Overview of method and its support tool for generating B from UML notations. Proceeding of 15th international conference on Automated Software En-gineering, Grenoble France (2000).

[18] Ledru, Y., Dupuy, S.: Expressing dynamic properties of static diagrams in Z. Conference of Approches Formelles dans l'Assistance au Développement de Logiciels, Rennes France (2003).

[19] Lenzerini , M., Nobili, P. (1990), "On the satisfiability of dependency constraints in Entity-Relationship schemata", Info. Syst. 90, Information Systems, volume 15, N° 4, 1990, p 453-461.

[20] OMG, editor: UML 2.0 Superstructure Specification. OMG (2005a). http://omg.org.

[21] Rochfeld, A., Negros, P.: Relationship of relationships and other inter-relationship links in ER model. Data and Knowledge Engineering 9 (1993) 205-221.

[22] Rumbaugh, J., Jacobson, I., Booch, G. UML 2.0 Guide de Référence, Edition Campus Press (2004).

[23] Soon-Kyeong, K., Carrington, D.: A formal mapping between UML models and Object-Z specifications. Proceedings of the First International Conference of B and Z Users on For-mal Specification and Development in Z and B. Springer, LNCS 1878 (2000) 2 – 21.

[24] Rational: http://www-306.ibm.com/ software/ rational/ sw-bycategory/ subcategory/ SW710.html

[25] Smith, J.M. et Smith, D.C.P. Database Abstractions: Aggregation and Generalization. In: TODS (1977). Vol. 2, Issue 2, pp. 105-133

[26] Sybase: http://www.sybase.com/products/information management/powerdesigner.

[27] Toby, J. T.: Database Modeling & Design. 3rd ed. Morgan, Kaufmann Series in data man-agement systems (1999).