Constraint Confrontation : An Important Step In View Integration

I. Comyn-Wattiau *, M. Bouzeghoub**

* ESSEC, Ecole Supérieure des Sciences Economiques et Commerciales, av B Hirsch, BP 105, 95021 Cergy-Pontoise Cedex, France
Tel : (33) 1 34 43 30 00 Fax : (33) 1 34 43 30 01 email : WATTIAU@FRESEC51

** Laboratoire MASI, Université Paris VI, Centre de Versailles, 45 av des Etats-Unis 78000 Versailles France

Abstract. This paper addresses the problem of constraint integration in database design. The approach is inserted in an incremental database design methodology supported by the design environment KHEOPS. The view integration step using semantic unification is followed by the initial constraint confrontation. The detection phase is a deductive process in which the contradictions and redundancies between constraints are exhibited. The following conflict resolution depends on the strategy initially chosen by the designer. Four strategies are presented and discussed. An example of use is given to illustrate the application of those strategies. This approach could be enlarged in an object oriented context where schemata would have not only constraints but also methods to compare.

Key-words. database design, view integration, integrity constraint, forward chaining, conflict resolution, deductive process.

1 Introduction

Initial views considered for integration generally contain a set of integrity constraints. Those constraints complete the description of objects and relationships between objects. The atomic objects, that is attributes, are characterized by a set of possible values, called domain. The link between a molecular object and each of its components can be described by cardinalities. The relations between the different constituents of a molecular object may be defined in terms of functional and multivalued dependencies. The relationships between different molecular objects are also submitted to cardinality constraints. Finally, other semantic constraints can be described in specific situations.

Most of papers about view integration address the problem of generating a global schema starting from views. Such a process needs the comparison of view constituents. Roughly, for each pair of objects, this comparison can lead to the generation of (i) one object by merging of the constituents if they are found identical or equivalent, (ii) two objects connected by an inclusion link if one is found as a subset of the other, (iii) two disjoint objects if they are found different. In that comparison, the constraints can be considered as one criterion, but this step must be followed by the confrontation of initial view constraints with the global schema constituents in order to complete the global schema.

When dealing with the constraints, the view integration literature talks more generally about the conflict resolution. Indeed, constraint integration assumes the resolution of conflicts consisting of contradictions and redundancies between the different constraints. The authors suggest several strategies for conflict resolution but they don't proceed to an automatic constraint integration [El-Masri & Wiederhold 79, Navathe & Gadgil 82, Batini & Lenzerini 84, Navathe & al 84, Mannino & Effelsberg 84, Spaccapietra & Parent 90, see Batini & al 86 for a survey]. None of these strategies is the best one but each one can be applied in a specific context, that is for instance database integration.

The main contributions to constraint integration are the algorithms producing third normal form relations from sets of functional dependencies [Beeri & Bernstein 79, Bernstein 76]. Such procedures are able to produce a minimal set of functional dependencies, thus pointing at the contradictions and eliminating the redundancies. But functional dependencies are not sufficient to represent all the constraints between data.

From a theoretical point of view, Convent has proved that conflictfreeness is undecidable [Convent 86]. More precisely, the problem of deciding whether a given set of integrity constraints logically implies a single constraint is undecidable. So computer-aided integration needs heuristics to support designers in detecting conflicts for restricted cases in which logical implication is easily provable.

This paper describes an investigation in such a direction. The approach is inserted in an incremental object-oriented database methodology [Bouzeghoub & Métais 91]. The constraint integration comes after the global schema generation using a semantic unification of initial views [Bouzeghoub & Comyn 90]. Its main originality is to use deductive techniques to provide a better formalization of integrity constraint knowledge.

The paper is organized as follows. In Section 2, the view integration process is described. Section 3 describes the constraint integration, dividing the problem into (i) the detection of conflicts and (ii) the resolution. Section 4 concludes this paper and gives directions for further research.

2 View Integration by Semantic Unification

Abstractly, the unification problem is the following [Knight 89] : given two descriptions x and y, can we find an object z that fits both descriptions ? The logical unification of two expressions, as that used in Prolog, consists in successive variable substitutions of one expression by corresponding constants of the other expression. For example, the unification of the two following terms: P(X,L(a,b),(Z,Y)) and P(a,f(Y,Z),(b,a)) leads to the following result: P(a,f(a,b),(b,a)) and P(a,f(a,b),(b,a)) using the following substitutions : replace X by a, L by f, Z by b and Y by a in the first expression and replace Y by a, Z by b in the second expression.

A view is a structure composed of complex objects. View integration consists in the comparison of their structures in order to build a schema including the two initial structures. This process can be formalized as a semantic unification of the view structures. The unification of two views consists in comparing element by element, with respect to their types (atomic or molecular objects), the components of the two views. However, the expected result is not necessarily the strict equality of the two structures but the detection of the overlapping parts and the disjoint parts. More precisely, given two concepts from two different views, we expect the view integration process to report whether these two concepts are equivalent, similar or dissimilar. To reach one of these results, instead of making substitution of variables as done in the traditional unification, we apply particular restructuring rules which permit in a finite set of transformations to decide whether the two compared concepts are equivalent or not. If the two terms are equivalent, they are merged into a single fact and represented only once in the resulting schema. If the two terms are dissimilar, they are both included in the resulting schema. If the two terms are neither equivalent nor dissimilar but present some similarity, if possible, we backtrack again to the unification process which will enhance the similarity or the dissimilarity by applying other restructuring rules or by interacting with the human designer.

The general principle of view integration is similar to that of the logical unification but the substitution rules of the latter one are replaced by restructuring rules in the former one. Indeed, as view structures have no variable, the unification proceeds by analogy or by deduction process. Matching two symbols mainly consists in comparing their natures (atoms, molecules, constraints, etc), their structures (lists of components), their constraints (domains, keys, dependencies, cardinalities, etc) and the represented populations (instances). The logical unification is inserted into the resolution principle whose objective is to prove or to refute a hypothesis. At each resolution step i, if the unification succeeds the step i+1 is proceeded, if the unification fails there is a backtrack to the step i-1 to choose another rule or to match against another expression.

The view unification is inserted into a methodological process whose objective is to reach a unified representation (called a conceptual schema) of different perceptions of the same universe of discourse. At each integration step, if the semantics of the conceptual schema is enhanced, the integration is validated and the following step is initiated. Otherwise the view to be integrated is considered either as redundant with respect to the referent schema or as badly defined with respect to the common understanding of the universe of discourse. Figure 1 shows the different steps of integration. The syntactic comparison tool examines the two view components using deduction by inheritance if necessary. If the first unification does not succeed, a view restructuring is operated. The second unification step is validated by interaction with the human designer. The result of this unification, i.e. the global schema, is completed by the constraint intégration.

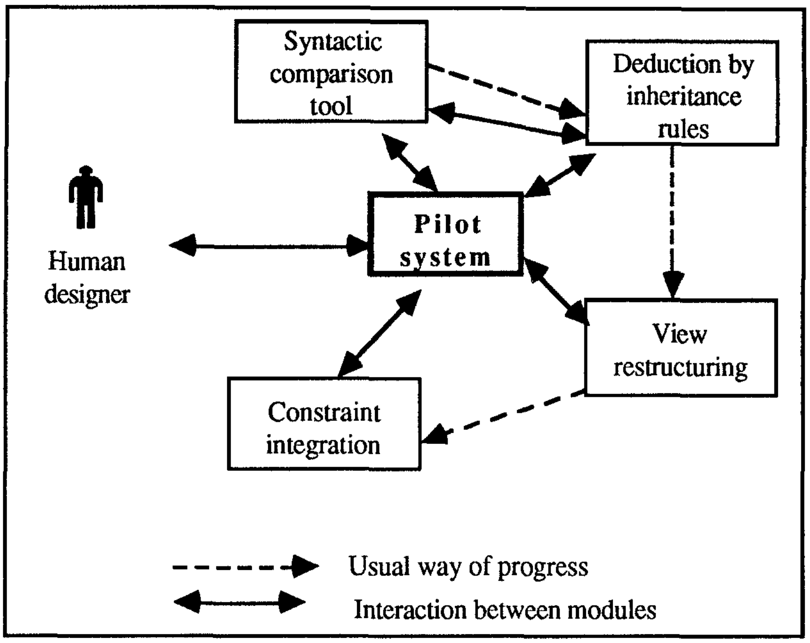


Fig 1. The view integration tool

The structural unification is based on three notions : the equivalence, the similarity and the dissimilarity.

(1) *Semantic equivalence between two concepts* : This notion depends on the nature of the two concepts to compare. Roughly, if we distinguish two levels of concepts, the basic level (atomic objects) and the structured level (molecular objects), we can intuitively define this semantic equivalence as follows :
* two atomic objects are equivalent if they have the same name and the same domain,
* two molecular objects are equivalent if they are composed of equivalent objects.

(2) *Semantic similarity between two concepts* : two given concepts (atomic or molecular) are similar if they have a same name, or a same domain, or a similar structure, or if they represent the same population. The similarity is not represented by a unique characteristic but by a set of properties which define a similarity degree whose value depends on the nature and the number of common properties. Then we consider the equivalence between two concepts as a similarity with its high degree, say value 1 if the similarity range is defined between 0 and 1.

(3) *Semantic dissimilarity between two concepts* : two concepts are dissimilar if they are neither equivalent nor similar. The dissimilarity between two concepts corresponds to a similarity with a lower degree, say 0 if the similarity range is defined between 0 and 1.

Consequently, we can represent the result of the unification of two concepts by a multiple information called similarity vector whose interpretation of the different components will vary from the equivalence to the dissimilarity. This vector is defined according to each type of concept. The differents types of concepts constitute a hierarchy starting from the most structured concept to the atomic concept. Then we distinguish two types of similarity vectors, corresponding to the two levels of objects :

$Sim_{structured\_object}(name, structure, constraints, population)$

$Sim_{atomic\_object}(name, domain)$

The *name* component specifies if the two names that are compared are identical or not. The *structure* component specifies whether the two compared concepts have same types or not. The type of an object consists in its list of components. The *constraint* component specifies if the two compared concepts have the same constraints or not. The *population* component specifies whether the two compared concepts represent the same set of instances or not. It concerns the intentional population. This component is only provided by the human designer who interacts with the view integration tool. The *domain* component specifies if two compared atomic objects have the same domain, included domains or different domains. Each of these components is a value ranging from 0 to +1. This value is assigned by the unification procedure. If this value is less than 1, the unification procedure generates a hypothesis (equivalence or dissimilarity depending on the value). If this hypothesis, say equivalence, is validated by the human designer, the component value becomes 1, otherwise (case of dissimilarity), the component takes the 0 value. Consequently, we have the following interpretation of the different components of the similarity vector :

- equivalence iff component = 1,
- similarity iff $0 <$ component < 1,
- dissimilarity iff component = 0,
- not considered iff component = '_ '.

The '_' symbol is used by analogy with the anonymous variable in Prolog.

Remark 1. Internally, when the designer is requested, we add 1 to the component (case of equivalence) or we subtract 1 from the component (case of dissimilarity). Hence at any time we know if the degree of similarity or dissimilarity is enhanced by the unification procedure or by the interaction with the human designer. At the end of the integration, the process may report the contribution of the human designer compared to the algorithmic decisions of the unification procedure.

Remark 2. This view integration process relies on one main heuristic, say 'if two objects have identical structures, they represent the same population'. Such a hypothesis is necessary to make the process more automatic.

A more detailed description of the unification procedure and mainly the evaluation of the similarity between views is given in [Bouzeghoub & Comyn 90].

3 Constraint Integration

We have defined earlier the role of constraints in the comparison of objects. These comparisons permit to enhance the similarity of the objects but they are not sufficient to infer the constraints associated to the global schema. The final step of integration consists in the constraint integration. The constraints which are considered are domains, keys, cardinalities and functional dependencies.

The domain constraints are considered at the time of the atomic constituent integration. So we consider here the integration of cardinalities, key and functional dependencies. It is relatively simple to detect a contradiction between two similar constraints, for example a contradiction between keys. If the subset A is a key for the object O in one view and the subset A is not a key for the same object in another view, the contradiction is explicit. However, it is necessary to detect contradictions between different constraints, that is for example between functional dependencies and keys. More generally, the conflict does not exist between only two constraints C and ¬C but inside a set of several constraints. For example, let E, A and B be objects belonging to both initial views, the set :

A is a key of E,
B is a monovalued constituent of E,
The functional dependency A->B is not valid.

contains a contradiction, but each pair of sentences is consistent.

To facilitate the expression of detection rules, we first describe the data model used. Then, the conflict detection process is described. The detection is followed by the resolution presented later.

3.1 The Data Model

The internal representation is the semantic model inspired by the different commonly used models (entity-relationship, extended entity-relationship, object models). Its content is easy to understand and allows a good representation of constraint interaction rules. We just describe the concepts used hereafter.

The atomic and molecular objects of views, namely attributes, entities and relationships, are the nodes of the semantic network. The arcs describe the semantic links between those objects. These arcs are labeled to express the constraints.

An arc '*p*' goes from a molecular object to each atomic constituent. Its label contains minimum and maximum cardinalities of the link. The inverse arc is called '*a*'. For example, *p*(E,A,[1,1]) means that each instance of the object E has exactly one value for its constituent A. *a*(A,E,[1,1]) means that each value of A corresponds to a unique instance of E.

The key constraint is expressed by the arc *key* : *key*(E,A) means that the set of atomic constituents A is a key of the object E. The functional dependencies are represented by '*fd*' arcs.

The constraint integration needs the representation of negative informations, for example it can be important to save the information about the fact that there is no functional dependency between two constituents. We use two arcs to express such negative information : *notkey*(E,A) means that A is not a key of the object E. *notfd*(E,A,B) means that, for the object E, the atomic constituent B is not functionally dependent on the subset A.

3.2 The Conflict Detection

The objective of view integration is to produce a minimal and consistent global schema. The minimality condition means that no redundancy may remain. The consistency condition means that any contradiction must be removed. The constraint integration operates as a deductive process aiming at the detection of contradictions and redundancies. The principle of this detection is described here.

The constraints contained in the initial views are added one by one to the global schema. They constitute the fact base. If the tool detects a contradiction between the fact base and the new constraint to be added, the conflict resolution step begins, leading to a possible fact base modification.

The rule base contains several types of rules. The first ones are Armstrong's axioms [Armstrong 74]. They formalize the interactions between functional dependencies. Let X, Y and Z be subsets of the set of atomic objects constituting the molecular object E :

RARM1 : If $X \supseteq Y$ Then fd(E,X,Y)
RARM2 : If fd(E,X,Y) Then fd(E, $X \cup Z$, $Y \cup Z$)
RARM3 : If fd(E,X,Y) and fd(E,Y,Z) Then fd(E,X,Z)

Another set of rules contribute to define the *notfd* and *notkey* predicates :

RNP1 : If not(fd(E,X,Y)) Then notfd(E,X,Y)
RNP2 : If not(key(E,X)) Then notkey(E,X)

A third set of rules formalize the interactions between all types of constraints. For example, let A and B be atomic constituents of E, RICC1 and RICC2 are two examples of interactions between keys and cardinality constraints :

RICC1: If key(E,A) Then p(E,A,[1,1])
RICC2: If key(E,A) Then a(A,E,[1,1])

The following RICD1 to RICD4 rules formalize part of the interaction between functional dependencies and cardinality constraints :

RICD1: If p(E,A,[_,1]) and fd(E,A,B) Then p(E,B,[_,1])
RICD2: If p(E,A,[_,1]) and $X \supseteq A$ and fd(E,X,B) Then p(E,B,[_,1])
RICD3: If p(E,B,[1,n]) and fd(E,A,B) Then p(E,A,[_,n])
RICD4: If fd(E,B,A) and a(A,E,[1,1]) Then a(B,E,[1,1]).

Others can be found in [Bouzeghoub 86]. Note that '_' is used to denote the 'unknown' value.

The inference mechanism uses a forward chaining technique. Before introducing the constraint C, the inference mechanism is applied to the set of consistent constraints. If C is deduced, then C is redundant and must not be inserted. If $\neg$C is deduced, then C is contradictory. Otherwise C can be added. The process ends when all the constraints have been considered.

3.3 The Resolution Strategies

The resolution of conflicts depends on the strategy chosen by the designer. We have defined four different strategies. In this section, we describe each one, giving in each case the possible use. Then the next sections will give an example and define the meta-rule which governs the use of those strategies.

The first three strategies have been found in the literature. The references and the suggested domains of application are given. The fourth one is original and can be seen as a step forward in the integration by generalization as defined by Dayal [Dayal & Hwang 84] and used in [Larson & al 89] to integrate attributes.

3.3.1 Predominant View Strategy

The first strategy consists in defining a preferred view. In case of conflicts, the preferred view is chosen. This strategy was suggested in [Navathe & Gadgil 82, Batini & Lenzerini 84]. In that case, the resolution process is made automatic by assigning preferences to views. Practically, the initial base fact contains the constraints associated to the preferred view. The other constraints are added one by one. If a contradiction or redundancy occurs, the constraint is not inserted.

This strategy has two main advantages : it makes the resolution an automatic process and it is very easy to implement. Such a strategy is applicable if both following conditions are valid. First, the preferred view is consistent. Secondly, the other views come as suggestions for improvement and thus can be partially rejected. Our opinion is that an incremental methodology database design meets these two conditions.

3.3.2 Interaction with the Designer

This strategy is completely opposite. It is based upon the idea that such a tool can be used to help the designer in the database schema generation. This help implies the detection of inconsistencies between the different designer declarations. These inconsistencies can be explained by the big size of the application or by the long lapse separating the beginning and the end of the process.

This strategy considers that contradictions are the results of wrong specifications. In case of such contradictions, the designer must be asked. So the conflict resolution is an interactive process in which the set of facts (including C) leading to a contradiction is presented to the designer who points at the wrong fact. If C is the wrong fact, C is rejected. If the wrong fact is already in the fact base, it is retracted and C is inserted.

An example of such dialog is given now. Suppose that the set of controlled constraint is defined by the four following constraints :

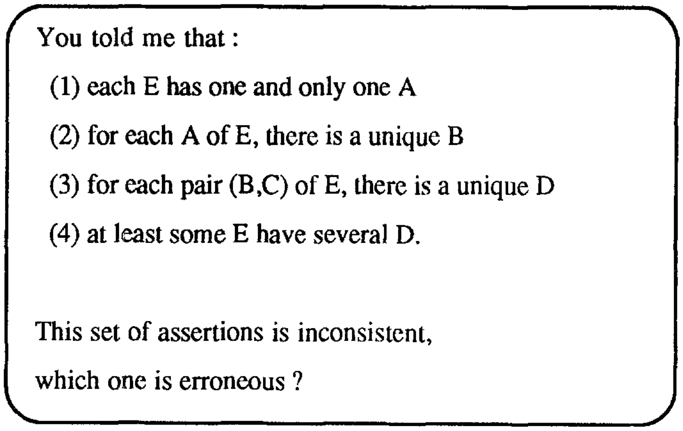
(1)	p(E,A,[1,1])	(A is a monovalued constituent of E)
(2)	fd(E,A,B)	(functional dependency from A to B inside E)
(3)	fd(E,BC,D)	(functional dependency from BC to D inside E)
(4)	p(E,C,[1,n])	(C is a multivalued constituent of E).

The constraint C now introduced is p(E,D,[1,n]). The forward chaining inference engine permits to deduce :

(5)	p(E,B,[_,1])	(deduced from 1 and 2 and RICD1)
(6)	p(E,D,[_,1])	(deduced from 3 and 5 and RICD2).

And C and (6) are inconsistent.

Practically, our tool paraphrases each constraint including C and proposes this list of sentences to the designer. The screen is displayed Figure 2. The designer points at the wrong sentence. The corresponding constraint is retracted from the fact base.



You told me that :

(1) each E has one and only one A

(2) for each A of E, there is a unique B

(3) for each pair (B,C) of E, there is a unique D

(4) at least some E have several D.

This set of assertions is inconsistent,

which one is erroneous ?

Fig. 2. Interaction with the designer

This solution can be used for database design to help the designer in exhibiting the conflicts between views. The tool is responsible for the consistency of the result, but the choices are made by the designer. It can be very useful when all the initial specifications are available but have been generated without control of an expert. Such a tool can also be used for computer assisted instruction.

3.3.3 Loose Constraining

Another strategy was suggested in several papers [Motro 87, Spaccapietra & Parent 90]. These authors propose to build a global schema meeting all the initial specifications. So, in each case of conflict, the less restrictive solution is chosen. Then, to take into account each initial view, the mappings between the global schema and the initial view express the possible restrictions. A few situations and the less restrictive choice in each situation are presented Figure 3.

Two main advantages can be presented to defend such a strategy. First, it makes an automatic resolution. Secondly, it does not alter the initial views. So it can be used in database integration where data already exist and where the global schema must reflect as best as possible the different local representations of data.

VIEW 1	VIEW 2	GLOBAL SCHEMA
Domain D1	Domain D2	Domain D1 $\cup$ D2
Minimal cardinality 0	Minimal cardinality 1	Minimal cardinality 0
Maximal cardinality 1	Maximal cardinality N	Maximal cardinality N
Key(E,A)	Notkey(E,A)	Notkey(E,A)
Functional dependency A -> B	Wrong functional dependency A -> B	Wrong functional dependency A -> B
Key(E,A)	Wrong functional dependency A -> B	Wrong functional dependency A -> B

Fig. 3. Contradictory specifications and resolution

3.3.4 Resolution by Specialization

The last strategy presented here is in keeping with the semantic modelling using generalizations and specializations [Smith & Smith 77]. The principle is to resolve the constraint conflict by specializing the object on which the contradiction takes place. This approach consists in declaring the contradictions between constraints as revealing overlapping class problems. In such a situation, the integration step must lead to a global schema where the different sub-classes with their own constraint sets coexist. Thus the resolution contributes to increase the automatic processing of class definition.

Practically, suppose that, for example, the object O1 in the first view, denoted View1(O1) and the object O2 in the second view, View2(O2) have been integrated in one unique object O in the resulting view. Suppose that two constraints C1 and C2 are respectively associated to O1 and O2 in the initial views. Four cases have to be considered :

1st case : *the constraints C1 and C2 are identical or equivalent*
formally C1=C2
In this case, C1 is added to the resulting view and associated to the object O.

2nd case : *the constraints C1 and C2 are contradictory*
formally C1 = $\neg$C2
The object O in the resulting view is specialized into two sub-classes O1 and O2. The constraints C1 and C2 are added to the resulting view and respectively associated to O1 and O2.

3rd case : *the constraints C1 and C2 are independant*
formally C1 $\neq$ C2 and C1 $\neq$ $\neg$C2
They are neither redundant (that is neither identical nor equivalent) nor contradictory. In

this case, the constraints C1 and C2 are added to the resulting view and both associated to O.

4th case : *the constraints C1 and C2 are compatible*
By compatible, we mean that one constraint is stronger than the other. For example in one view, one constituent of O is mandatory, in the other view it is optional. To express this compatibility, we have defined an order. This order is total inside a set of compatible constraints. It is defined as follows :

C1 > C2 if C1 is stronger than C2.

In this case, the object O in the resulting view is specialized in one sub-class O1. The constraints C1 and C2 are added to the resulting view. The stronger constraint, say C1, is associated to the sub-class O1. The other constraint, say C2, is associated to O. To illustrate our definition of order, Figure 4 enumerates a few cases of what we call constraint compatibility.

CONSTRAINT C1	CONSTRAINT C2
The domain of A is D1	The domain of A is D2 $\supset$ D1
The set of attributes E1 is a key	The set of attributes E2 $\supset$ E1 is a key
The set of attributes E is a key	The set of attributes E is not a key
There is a functional dependency from A to B	There is no functional dependency from A to B
The set of attributes E is a key	There is no functional dependency from E to any other constituent F
The attribute A is monovalued	The attribute A is multivalued
The attribute A is mandatory	The attribute A is optional
The minimum cardinality of entity E in relationship R is 1	The minimum cardinality of entity E in relationship R is 0
The maximum cardinality of entity E in relationship R is 1	The maximum cardinality of entity E in relationship R is n

Fig. 4. Compatible constraints where C1 > C2

The previous discussion was about the comparison of two constraints. But the integration process consist in adding one by one the constraints to a controlled set. So we give now the algorithm used for the introduction of a constraint C coming from the initial view Viewi and associated to the object which led to the integrated object O. The algorithm called CONSTRAINTINTEGRATION(Viewi, C, O) looks for possible constraints already inserted and contradictory or compatible with C. If such a constraint exists, the constraint C is associated to the object O or to one of its sub-classes. Figure 5 gives the algorithm. Figure 6 describes the subroutine BIND(C, O, Viewi) which associates the constraint C to the sub-class of O corresponding to the view Viewi.

```
CONSTRAINTINTEGRATION(Viewi, C, O)
BEGIN

IF Resultingview(O,C',Viewj) and C=C'
/* in the resulting view, a constraint C' identical or
   equivalent to C is associated to O */
  THEN
    REJECT C
  ELSE
    IF Resultingview(O,C',Viewj) and C'<C
     /* in the resulting view, a constraint C' compatible with C
     but less strong than C is associated to O */
    THEN
      BIND(C,O,Viewi)
    ELSE
      IF Resultingview(O,C',Viewj) and C'>C
       /* in the resulting view, a constraint C' compatible with C
       but stronger than C is associated to O */
      THEN
       /* associate C' to the sub-class of O corresponding the view Viewj */
        BIND(C',O,Viewj);
        RETRACT Resultingview(O,C',Viewj);
       /* associate C to O in the resulting view */
        INSERT Resultingview(O,C,Viewi)
      ELSE
        IF Resultingview(O,C',Viewj) and C' ¬= C
       /* in the resulting view, a constraint C' inconsistent
          with C is associated to O */
        THEN
          BIND(C,O,Viewi);
          BIND(C',O,Viewj);
          RETRACT Resultingview(O,C',Viewj)
        ELSE
         /* all the constraints already associated to O
            are independent of C */
          INSERT Resultingview(O,C,Viewi)
        ENDIF
      ENDIF
    ENDIF
ENDIF
END
```

Fig. 5. The algorithm

```
BIND(C, O, Viewi)
/* associates the constraint C to the sub-class of O
    coming from the view Viewi
  and creates this sub-class if not done */
BEGIN
IF Resultingview(s(O,Viewi,Oi))
  /* in the resulting view, the object O has a sub-class Oi matching
     with the correspondant object in Viewi */
THEN
   /* inserts the constraint C in the resulting view,
   associated to the object Oi */
 INSERT Resultingview(Oi,C,Viewi)
ELSE
   /* creates the sub-class of O corresponding to Viewi */
   INSERT Resultingview(s(O,Viewi,Oi));
   /* inserts the constraint C in the resulting view,
   associated to the object Oi */
   INSERT Resultingview(Oi,C,Viewi)
ENDIF
END
```

Fig. 6. A procedure to associate a constraint to an object in the resulting view

3.4 Example

This sections gives a very simple example of two views with different constraints. We show how each previous strategy applies in this case. The example describes the involvement of employees in projects. In the first view, each employee is involved in one and only one project. In the second view, an employe may participate in several projects. The two views are graphically described Figure 7. Suppose that the first view is the preferred view. The integration will lead to forbid employes several involvements. If the strategy is loose constraining, the second view is chosen. In case of resolution by specialization, the employees are allowed to participate in several projects, except a sub-class of them who are bordered on a unique project. By interaction with the designer, the two options are suggested. The four possible results of integration are summmed up Figure 8.

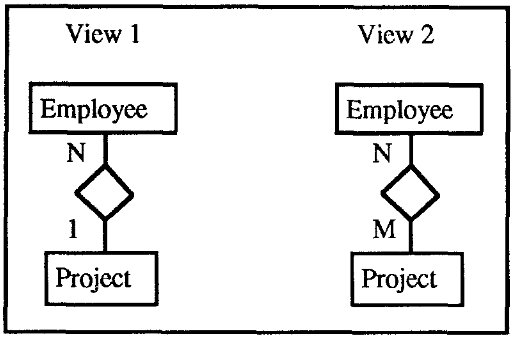


Fig. 7. The two initial views

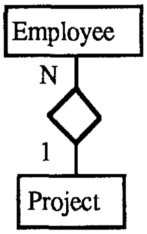
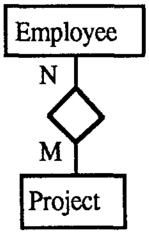
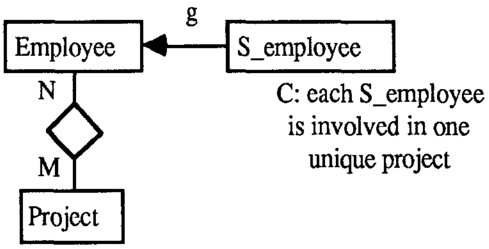
Predominant view strategy 	*Interaction with the designer* You told me that : (1) Each employee is involved in one project (2) An employee may be involved in several project. This set of assertions is inconsistent, which one is erroneous ?
Loose constraining 	*Resolution by specialization*  Employee g S_employee N M Project C: each S_employee is involved in one unique project

Fig. 8. Application of four strategies

3.5 Strategy Selection

For each strategy, we have defined above the different conditions of application and the advantages. This analysis allows us to describe now the decision support module which helps the designer in the choice of the best strategy.

The first and most important criterion to take into account is the initial specification reliability. Without this condition, no automatic resolution can be considered. The detection is made by the tool and the resolution emerges from a dialog with the designer. If the initial specifications are reliable, it is important to know if they are modifiable or not. In database integration for example, initial specifications represent real local databases and thus cannot be altered. Finally, the fourth strategy can be used if the data model allows the expression of specializations. All these considerations are summed up in the decision process graphically drafted Figure 9.

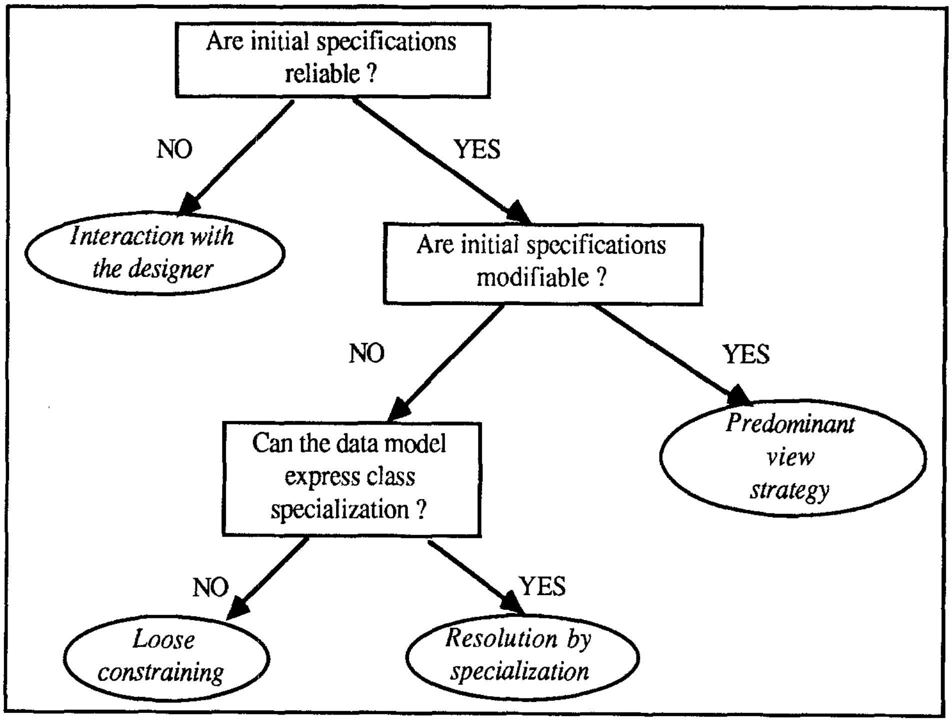

Are initial specifications reliable ?
NO
YES
Interaction with the designer
Are initial specifications modifiable ?
NO
YES
Can the data model express class specialization ?
Predominant view strategy
NO
YES
Loose constraining
Resolution by specialization

Fig. 9. The strategy selection decision tree

4 Conclusion

In this paper we have presented the principle of a deductive process for constraint integration. Our approach of constraint integration is added at the end of the view integration process to complete the resulting view. We have developed and implemented a view integration technique based on a structural unification of initial views [Bouzeghoub & Comyn 90]. Constraint integration comes after that to detect and solve the contradictions and redundancies between initial specifications. This constraint integration, as the view integration process, is to be considered as a decision support system which helps the designer in giving him information about similarities or contradictions. The final integration relies on the designer's decision.

The detection is a deductive process using a set of interaction rules modelling the relations between constraints. Key, functional dependencies, cardinalities and domains are considered. Some negative information is treated, thus making the technique applicable to incremental database design in which the global schema is obtained by successive refinements of an initial specification.

The conflict resolution depends on the strategy chosen by the designer. Four strategies have been defined and discussed. The last one, called resolution by specialization, is original. In case of conflict, it suggests to specialize the object into sub-classes in which constraints are no more contradictory.

We feel that this approach could also be enlarged to take into account integration problems in an object oriented context. That is an important issue which still remains to be investigated. Another interesting extension is to consider general integrity constraints and to confront our constraint integration methodology with such constraints.

References

[Armstrong 74] "Dependency Structures of Database Relationships", Armstrong W, Proceed. 1974 IFIP Congress, North-Holland, Amsterdam, 1974.

[Batini & Lenzerini 84] "A Methodology for Data Schema Integration in the E-R Model", Batini C, Lenzerini M, IEEE Transactions on Software Engineering, Nov 84.

[Batini & al 86] "A Comparative Analysis of Methodologies for Database Schema Integration", Batini C, Lenzerini M, Navathe SB, ACM Computing Surveys, Dec 86.

[Beeri & Bernstein 79] "Computational Problems Related to the Design of Normal Form Relational Schemas", Beeri C, Bernstein P, ACM Transactions on Database Systems, March 79.

[Bernstein 76] "Synthesizing third normal form relations from functional dependencies", Bernstein P, ACM Transactions on Database Systems, 1976.

[Bouzeghoub 86] "SECSI : Un Système Expert en Conception de Systèmes d'Information", Bouzeghoub M, Thèse de doctorat de l'université Paris VI, March 1986.

[Bouzeghoub & Comyn 90] "View Integration by Semantic Unification and Transformation of Data Structures", Bouzeghoub M, Comyn I, Proceed of the ER conf, Lausanne, 1990.

[Bouzeghoub & Métais 91] "Semantic Approach For Object Oriented Database Design", Bouzeghoub M, Métais E, Proceed of the VLDB conf, Barcelone, 1991.

[Convent 86] "Unsolvable Problems Related to the View Integration Approadch", Convent B, ICDT Conf, Roma, Sept 1986.

[Dayal & Hwang 84] "View Definition and Generalization for Database Integration in a Multidatabase System" Dayal U, Hwang H, IEEE Transactions on Software Engineering, Nov 1984.

[Elmasri & Wiederhold 79] "Data Model Integration Using the Structural Model" Elmasri R & Wiederhold G, Proceed. ACM-SIGMOD internat. conf., Boston 1979.

[Knight 89] "Unification : A Multidisciplinary Survey" Knight K, ACM Computing Surveys, 1989.

[Larson & al 89] "A Theory of Attribute Equivalence in Databases with Application to Schema Integration", Larson JA, Navathe SB, El-Masri R, IEEE Transactions on Software Engineering, Vol 15(4), 1989.

[Mannino & Effelsberg 84] "Matching Techniques in Global Schema Design", Mannino MV, Effelsberg W, Proceed. IEEE COMPDEC conf., Los Angeles, CA, 1984.

[Motro 87] "Constructing Superviews", Motro A, IEEE Transactions on Software Engineering, July 1987.

[Navathe & Gadgil 82] "A Methodology for View Integration in Logical Database Design", Navathe SB, Gadgil S, Proceed. 8th VLDB Conference, 1982.

[Navathe & al 84] "Relationship Merging in Schema Integration", Navathe S, El-Masri R, Sashidar T, Proceedings 10th VLDB Conference, 1984.

[Smith & Smith 77] "Database Abstractions : Aggregation and Generalization", Smith JM, Smith DCP, ACM Transactions on Database Systems Vol 2, N° 2, June 1977.

[de Souza 86] "SIS- A Schema Integration System", de Souza JM, Proc. BNCOD 5 conference, 1986.

[Spaccapietra & Parent 90] "View integration : A step forward in solving structural conflicts", Spaccapietra S, Parent C, Proc. of VIes Journées Bases de Données Avancées, Montpellier, Sept 1990.

[Tucherman & al 86] "A Software Tool for Modular Database Design", Tucherman L, Furtado AL, Casanova MA, Proceed. of 11th VLDB Conf, Stockholm, Aug 1985.