Schema evolution in CASE databases

M. Bouneffa & N. Boudjlida
CRIN, Batiment LORIA, Campus scientifique, BP 239, 54506 Vandoeuvre-les-nancy, France

Abstract. In this paper we present an approach of CASE database schema evolution. We also define a set of schema quality criteria and prove that the evolution process preserves them. Schema evolution is performed by a schema integration process based on assumption management. Before integrating two schemas, relationships or correspondences relating their concepts must be identified. Such task is called schema comparison. To do this, our approach combines schema transformation with an interactive process which generates and validates hypothetic correspondences or assumptions. The data schema being viewed as a multi-graph, we classify and apply quality criteria to every node in the graph (*local criteria*) as well as to the whole graph (*global criteria*).

1 Introduction

New applications of databases such that CAD (Computer Aided Design), CAM (Computer Aided Manufacturing) and CASE (Computer Aided Software Engineering) generate and manipulate data with a large variety of structures and behaviors[1]. Indeed, defining rich semantic models is necessary for representing these kinds of data. Moreover, the experimental and iterative nature of these applications intensify the need of evolutivity and flexibility at the data definition and manipulation levels.

Evolving a database schema consists in modifying its definitions and propagating these changes to the database instances and programs. This evolution must preserve the consistency of the database schema. Much research has been devoted to this problem, however in general they achieve the evolutivity of schema by a set of atomic operations and using a set of consistency maintaining rules[5][12][9]. Despite its gain of flexibility, this approach presents the disadvantage of keeping in charge of the designer the responsibility of making the evolution plan which is also error prone. This methodological poorness makes it harder the construction of CASE databases for which the complexity is one of the main features.

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

The main contribution of this paper consists in proposing an evolutive CASE databases construction process. The process is usable in the initial database design or during the database life. In software engineering, quality criteria, like cohesion or coupling, can be used as a measurement of the design. To our knowledge, a few work is done in the measurement of the data structure quality. In database applications, the quality of the database schema is at least as important as the quality of the programs that act on the database. In this context, we state some quality criteria to which a database schema must conform. We show that the quality criteria are preserved by the evolution process. The approach is essentially based on a schema integration mechanism which is based on an assumption management process.

This paper is organized as follow: section 2 situates our approach in the respective areas of both schema evolution and software quality management. Section 3 introduces a semantic and object-oriented data model we used in our approach. In sections 4 and 5 we describe with more details the schema integration process. We focuses the description on the assumption management. A detailed example showing the schema integration process is then given at the final of section 5. Section 6 sets the schema quality criteria and describes formally how the integration approach preserves them. In the conclusion (section 7) we summarize the results of the paper and suggest possibilities for further research.

2 Related work

Schema integration is defined as *the activity of integrating the schemas of existing or proposed databases into a global, unified schema* [2]. Schema or view integration may also, be seen as an evolution process. Starting from two or more source schemata, an integrated schema is constructed. This is done by choosing one schema as the target schema, and adding the others to it[10]. Much research have been devoted to this area. A comprehensive survey of schema integration methods can be found in [2]. The prevalent approach in schema integration is to derive an integrated schema by merging equivalent concepts. The most important activity is then, to establish correspondences or relationships between these concepts. To do this, some approaches consist of using name or types comparison (i.e. syntactic approach). Other approaches use probabilistic knowledge (i.e. more semantic) [11]. The most important limitation of these approaches is that they are, in general, incapable of expressing some relationships between two semantically equivalent but structurally different constructs. The establishment of such relationships needs to transform schemas to be integrated before to compare them.
We propose an interactive process to integrate schemas on the basis of syntactic and semantic knowledge. To avoid the limitation of both syntactical and semantical approaches, our process includes schema transformation as a part of schema comparison. This process begins by generating hypothetic relationships or correspondences which must be validated in a an efficient manner.

In database applications the quality of the database schema is, at least, as important as the quality of the programs that are designed to act on the database. In this framework, quality criteria like coupling and cohesion of a software design can be applied to the

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

programs but only few work has concentrated on the quality of the database schema. Some subjective criteria, like readability of a schema, can be stated but they are not easily measurable. Schema quality criteria can be derived from three standpoints. The first one is the underlying data model. Indeed, every data model has inherent constraints that can lead to quality criteria. For instance, the (initial) relational data model theory [4] imposes that every attribute within a relation has an atomic domain. It also imposes to distinguish key attributes. This kind of constraints leads to reject ill-defined schema design or part of a it.

The second standpoint from which design criteria can derive is the use of the data model abstraction mechanisms. Indeed, constraints can be fixed on the iterative usage of the abstraction concepts in the design phase. As an example,[6] presents a *complete family* of rules that ensures irredundancy, unambiguity and satisfiability of the object types that compose the data schema.

Finally, the third standpoint from which quality criteria may be considered, is the application domain. This standpoint is obviously more difficult to be formalized since it is human-dependent.

Thus, the data schema being viewed as a multi-graph, we classify and apply quality criteria to every node in the graph (these criteria are called *local criteria*) as well as to the whole graph (*global criteria*). Local criteria mainly derive from constraints that are inherent to the data model, while global criteria restricts the use of the successive use of the abstraction constructs that are offered by the data model. These are presented in section 6 and we demonstrate, in the same section, that the integration and the evolution process preserves them. The forthcoming section presents the data model concepts we use in our approach.

3 The data model

The data model encompasses basic types of objects as well classical semantic constructors and relationships among objects. The concepts of the data model are briefly exposed hereafter.

3.1 Informal presentation

To present the data model, we first describe the object types concept as well as the different constructors. We then explain the notion of schema.

3.1.1 Object types

There are three kind of object types (shown in the figure 1).

1. **Basic types** are generally used for Input/Output applications. Such types may be atomic (integers, boolean, string, ...) or structured like arrays, lists or tuples of atomic types. Basic types are comparable to data types of structured programming languages.

2. **Entity types** would be perceived as entities in the Entity/Relationship model[3] or abstract types in semantic data models [6][8]. Entity types are basically used to represent physical and conceptual objects of the real world. Thus, an entity type may represent, for instance, a module and hardware or software resources in a programming environment.

3. **A relationship type** represents a semantical association between two entity types (binary relationships). Such associations may explain dependencies like those existing between software modules (e.g. export/import of resources). The two entity types associated by a relationship type are called *participants* and labeled respectively by two string values representing their roles.

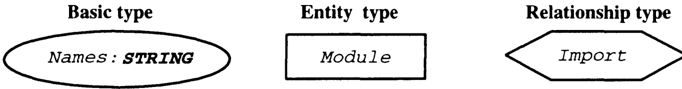

Figure 1: Object types Examples

3.1.2 The constructors

We have defined three constructors :

The attribute constructor can be viewed as a function defined between : two entity types; an entity type and a basic type or a relationship type and a basic type. An attribute is defined by a name which is a string value and the two object types representing its range and domain. For instance, in the figure 4 the attribute *author* has as a name *author*, as a range the entity type *Module* and as a domain the basic type *STRING*.

The composition relationships or links defined between two entity types can be viewed as a combination of classical aggregations (tuples) and grouping[6]. The figure 2 shows a module as an aggregation of both a specification and many bodies (or implementations). A composition link is defined by giving a composite entity type (e.g. *Module* in the figure 2) and the components which are designed by their roles or names and their cardinalities. A multiple cardinality of a component can be considered as a set or grouping constructor of this component.

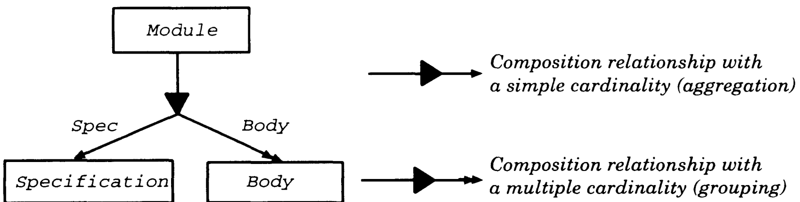

Figure 2: Composition constructors examples

The subtyping relationship or link defined between two entity types is the same as classical IS_A relationship[6] restricted to a simple inheritance.

Remark: Entity, relationship and basic types encapsulate the definition of operations or methods which are the unique way by which their instances can be manipulated.

3.1.3 Schemas

A schema is a set of entity, relationship and basic types related by subtyping, composition and attributes relationships. A schema defines also a set of integrity constraints which

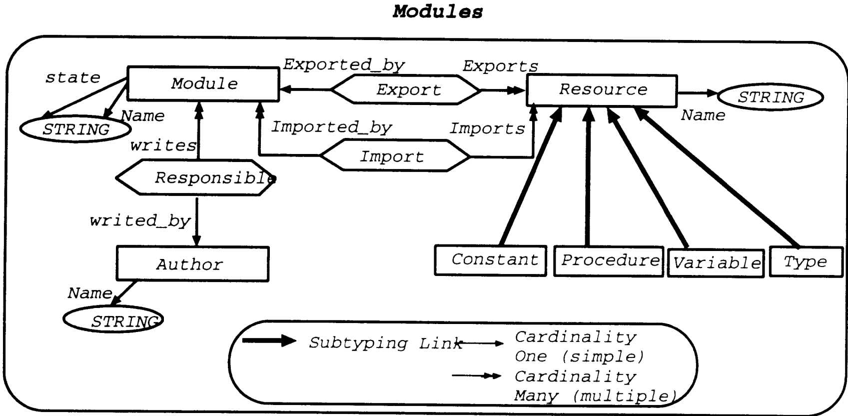

Figure 3: Modular programming environment : in the-large view

must be verified by the database instances. The figures 3 and 4 shows the schemas *Modules* and *Mod – Comp* which represent "in-the-large" and "in-the-small" views of a modular programming environment. These two schemas will be used to illustrate our schema integration process.

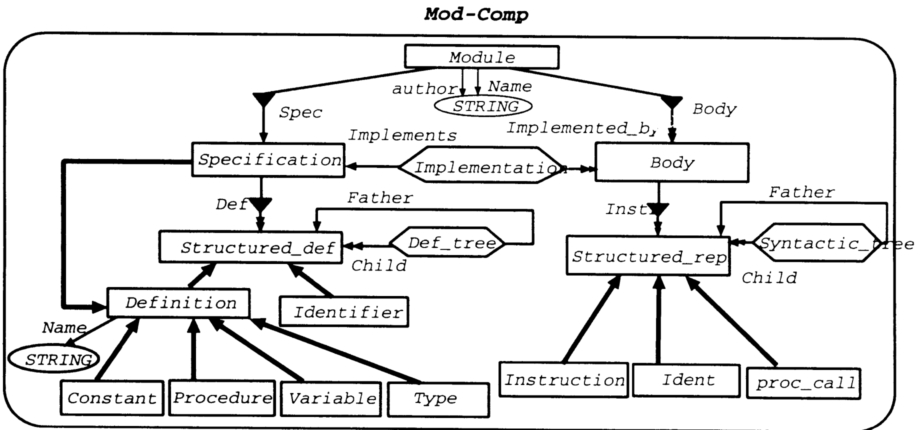

Figure 4: Modular programming environment : in-the-small view

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

3.2 Formal definition of a data schema

A schema *Sch* is described by the tuple $< Ent\ ;\ Bas\ ;\ Rel\ ;\ CONSTR\ ;\ SUB\ ;\ COMP >$.

- Ent, Bas, Rel and $CONSTR$ are respectively the sets of entity types, basic types, relationship types and constraints defined in the schema.
- SUB and $COMP$ are the subtyping and composition hierarchies: $SUB = [Ent\ ;\ <_S]$ and $COMP = [Ent\ ;\ <_C]$, where: $<_S = \{< e_i\ ;\ e_j >\} \subseteq Ent\ X\ Ent$ and $<_C = \{< e_i\ ;\ e_j >\} \subseteq Ent\ X\ Ent$.

The subtyping relationship (denoted by $<_S^*$) is defined as following:

$$\forall e_1 \in Ent\ \forall e_2 \in Ent,\quad e_1 <_S^* e_2\ (e_1\ is\ a\ subtype\ of\ e_2) \implies$$
$$(\exists t,\ t \in <_S\ \wedge\ t = < e_1\ ,\ e_2 >) \vee (\exists e_3,\ e_3 \in Ent\ such\ that\ < e_1\ ,\ e_3 > \in <_s\ \wedge e_3 <_S^* e_2)$$

The composition relationship (denoted by $<_C^*$) is defined in the same manner.

4 The schema integration process

In this section, we describe the schema integration process we developed in order to implement our schema evolution approach. This process produces a schema *Sch* by merging two other schemas Sch_1 and Sch_2 which represent two partial views of a database. The schema *Sch* can then be considered as an evolution of Sch_1 and/or Sch_2.
In order to achieve the integration of two schemas, three important activities must be performed (figure 5): *comparison*, *conforming* and *merging* of these schemas.

The aim of schema comparison is to detect common concepts and check all conflicts in representing them. Such conflicts may be *name* conflicts (homonymous, synonymous) or structural conflicts. A structural conflict may exist when a common concept is represented by incompatible data structures. For instance an *author* can be represented by a string value or a more structured entity. The comparison of two schemas allows also the checking of semantic links like subtyping between two types belonging to two different schemas.

The conforming of two schemas consists of eliminating all conflicts detected by the schema comparison. *Name* conflicts can be eliminated by simple renaming operations whereas structural conflicts requires the performing of data structure transformations.

Both comparison and conforming of schemas produce two schemas ready to be merged. The merging operation consists in superposing data types representing common concepts. The rest of the two schemas are then related to the common types by means of the interschema semantic links.

The remainder of this paper presents essentially the schema comparison process. A Detailed presentations of both schema transformations and merging can be found in [7] and [2].

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

5 The Schema comparison

The comparison of two schemas consists of building a correspondence between their components. This correspondence is materialized by a set of tuples $< cl; c_1; c_2 >$. Such a tuple means that the concepts c_1 and c_2 belonging to two different schemas are related by the correspondence link cl which may be a similarity link or an interschema semantic link like subtyping relationship. The construction of this correspondence is performed in two steps (Figure 5). We first generate a set of hypothetic links extracted from knowledge included in the schemas to be integrated and possibly augmented with additional information given by the designer. These links are assumptions and their validation may need the designer's intervention. In order to efficiently manage the interaction with the designer we have established the function *choose a link for validation* (figure 5) that relies on a priority-order on assumptions. A high priority is given to those assumptions that validate the greatest number of remaining assumptions.

Sch1
Sch2
Comparison
Generate the initial set LH of hypothetic interschemas correpondence links
Ask more information to the designer
Yes
LH = ø
No
Choose a link *lh* to validate
Conflict?
No
Confirm or refute *lh* and update a LH
Yes
LH = ø
No
Schema conforming
Transform Sch1 and/or Sch2
Yes
Schema merging
Merge Sch1 and Sch2 and produce Integ(Sch1,Sch2)
STOP

Figure 5: The schemas integration process

The remainder of this section describes first the rules used to generate hypothetic

correspondence link (assumptions). We then explain the validation process of these assumptions by giving a detailed example which shows the integration of the two schemas depicted by the figures 3 and 4 ($Modules$ and $Mod - Comp$).

5.1 Rules of Assumption generation

Information which makes possible the assumption generation is extracted basically from the definition of the data structures. In addition to this syntactical information, the designer can provide semantical knowledge represented by additional correspondence links. We defined three categories of assumptions generation rules concerning three kind of links:

- similarity of two compatible concepts
- similarity of two incompatible concepts
- the generation of interschema subtyping relationships.

Compatible concepts are types or constructors with the same nature. For instance, two entity types are compatible when an entity type and a relationship type are not compatible. We can note that a similarity of two incompatible concepts is in fact a structural conflict.

Before defining explicitly the rules of the assumption generation, we define the following three sets LH, LC and LI which represent respectively : the *hypothetic*, *valid* and *refuted* links. These three sets will be used to show a formal description of the assumption generation rules.

5.1.1 Similarity of two compatible concepts

1. **Rule 1: The similarity of two attributes**
   Let t_1 and t_2 be two entity types respectively defined in two different schemas Sch_1 and Sch_2. Two attributes a_1 and a_2 respectively defined in t_1 and t_2 are assumed to be similar if:

   - they have the same name or the same domain, or
   - their domains are similar or assumed to be similar (in the case where their domains are not basic types).

   In addition, to avoid the generation of useless assumptions, we have to verify that this link was not validated or refuted and the types t_1 and t_2 were not recognized as distinct. We do the same verification for all rules defined bellow.

   This can be formalized by:

   Rule 1 : $<$ **Sim_attr** ; $t_1.a_{1,Sch_1}$; $t_2.a_{2,Sch_2}$ $>\in LH$ if:
   $(a_1 = a_2) \vee (Domain(a_1) = Domain(a_2)) \vee$
   $(<$ **Sim_ent** ; $Domain(a_1)$; $Domain(a_2)$ $> \in LH \cup LC)) \wedge$
   $(<$ **Sim_attr** ; $t_1.a_{1,Sch_11}$; $t_2.a_{2,Sch_2}$ $> \notin LC \cup LI) \wedge$
   $(<$ **Sim_ent** ; t_{1,Sch_1} ; t_{2,Sch_2} $> \notin LI)$

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

2. **Rule 2: The similarity of two components**
   Two components c_1 and c_2 defined on two entity types t_1 and t_2 respectively in two schemas Sch_1 and Sch_2 are assumed to be similar ($< \textbf{Sim_comp} ; t_1.c_{1,Sch_1} ; t_2.c_{2,Sch_2} > \in LH$) if, they share the same name, or their types are similar or assumed to be similar.

   Similar rule can be used to generate similarity assumption between two roles r_1 and r_2 defined on two relationship R_1 and R_2. In this case the correspondence is symbolized by the triplet: $< \textbf{Sim_role} ; R_1.r_{1,Sch_1} ; R_2.r_{2,Sch_2} >$.

3. **Rule 3: The similarity of two entity types**
   Two entity types are assumed to be similar ($< \textbf{Sim_ent}; t_{1,Sch_1}; t_{2,Sch_2} > \in LH$) if:

   - they have the same name, or
   - they share at least two attributes or components which are similar or assumed to be similar, or
   - they are both components of two other entity types which are similar or assumed to be similar.

4. **Rule 4: The similarity relationship type**
   Two relationship types are assumed to be similar ($< \textbf{Sim_rel} ; R_{1,Sch_1} ; R_{2,Sch_2} > \in LH$) if:

   - they have the same name, or
   - they share at least two attributes or roles which are similar or assumed to be similar.

5.1.2 Similarity between two incompatible concepts

The correspondence links represented by the triplets:
$< \textbf{Sim_bas_ent} ; t'.a ; t >$, $< \textbf{Sim_bas_rel} ; t'.a ; r >$ and
$< \textbf{Sim_ent_rel} ; t ; r >$, $< \textbf{Sim_comp_rel} ; t'.c ; r >$
denote respectively the facts that an entity type is similar to an attribute (which domain is of basic type), a relationship type is similar to an attribute, an entity type is similar to a relationship and a composition link is similar to a relationship. These triplets which can be generated by name equality, denote structural conflicts in the representation of a same real world concept.

5.1.3 The detection of interschema subtyping relationship

An entity type t_1 defined in a schema Sch_1 is assumed to be a subtype of another entity type t_2 defined in another schema Sch_2 if:

- inherited properties of t_1 are similar or assumed to be similar to properties of t_2 or,
- own properties of t_1 are similar or assumed to be similar to those of t_2 and t_2 have at least one subtype in Sch_2.

As an example, the entity type *Resource* defined in the schema *Modules* (figure 3) may be assumed as a subtype of the entity type *Definition* defined in $Mod-Comp$ (figure 4) because the attributes *name* of both *Ressource* and *Definition* are assumed similar and *name* is passed on by *definition* to its subtypes.
The generation of an interschema subtyping relationship may introduces conflicts in the resulting subtyping hierarchy (cycles, multiple inheritance ...). These kind of conflicts and their resolution are described in the complete example (section 5.2) and in the section 6.

5.2 The assumption validation process

Assumptions generated in the above section must be confirmed or refuted. Such an operation is called assumptions validation process and may amounts a supply of information about the real world semantic of data. The aim of this activity is to realize this process optimizing the interaction with the designer. Assumptions which introduce conflicts have the higher priority since it is often not possible to continue the schema comparison without solving them. In the lack of conflicts the first assumptions to be validated are those which have as arguments the most structured types (composite types etc...). The validation of an assumption often leads to an automatic validation (refutation or confirmation) of other assumptions. Such propagations have been showed in the example given below :

5.2.1 An example

This example presents an application of our schemas integration method in a CASE databases design context. It shows the integration of schemas **Modules** and **Mod-Comp** (figures 3 and 4) which depict respectively a coarse grain and a fine grain representations of a modular programming environment.

The integration process generates first the initial set (LH) of hypothetic correspondence links or assumptions: $LH = \{$

(1) $< Sim_attr \; ; \; Module.name_{Modules} \; ; \; Module.name_{Mod-Comp} >$,
(2) $< Sim_attr \; ; \; Module.name_{Modules} \; ; \; Definition.name_{Mod-Comp} >$,
(3) $< Sim_attr \; ; \; Resource.name_{Modules} \; ; \; Module.name_{Mod-Comp} >$,
(4) $< Sim_attr \; ; \; Resource.name_{Modules} \; ; \; Definition.name_{Mod-Comp} >$,
(5) $< Sim_attr \; ; \; Author.name_{Modules} \; ; \; Module.name_{Mod-Comp} >$,
(6) $< Sim_attr \; ; \; Author.name_{Modules} \; ; \; Definition.name_{Mod-Comp} >$,
(7) $< Sim_ent_attr \; ; \; Author_{Modules} \; ; \; Module.author_{Mod-Comp} >$,
(8) $< Sim_ent \; ; \; Module_{Modules} \; ; \; Module_{Mod-Comp} >$,
(9) $< Sim_ent \; ; \; Constant_{Modules} \; ; \; Constant_{Mod-Comp} >$,
(10) $< Sim_ent \; ; \; Procedure_{Modules} \; ; \; Procedure_{Mod-Comp} >$,
(11) $< Sim_ent \; ; \; Variable_{Modules} \; ; \; Variable_{Mod-Comp} >$,
(12) $< Sim_ent \; ; \; Type_{Modules} \; ; \; Type_{Mod-Comp} >$,
(2) $\Longrightarrow$ (13) $< Sim_ent \; ; \; Module_{Modules} \; ; \; Definition_{Mod-Comp} >$,
(3) $\Longrightarrow$ (14) $< Sim_ent \; ; \; Resource_{Modules} \; ; \; Module_{Mod-Comp} >$,
(4) $\Longrightarrow$ (14) $< Sim_ent \; ; \; Resource_{Modules} \; ; \; Definition_{Mod-Comp} >$,
(4) $\Longrightarrow$ (16) $< Subtype \; ; \; Resource_{Modules} \; ; \; Definition_{Mod-Comp} >$,

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

(4) $\Longrightarrow$ (17) $< Subtype \ ; \ Definition_{Modules} \ ; \ Resource_{Mod-Comp} >$,
}
So, $LH = \{(1)\ldots(17)\}$
The second stage consists on validating the above assumptions, to do this we begin by consider those which introduce conflicts. Such assumptions are the assumption (7) which denotes a similarity between an entity types and a basic type attribute and assumptions {(9),(10),(11),(12),(16),(17) } which introduce a cycle and multiple inheritance in the subtyping hierarchy (figure 6).

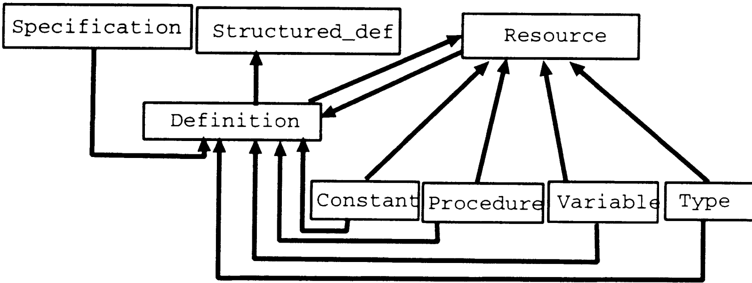

Figure 6

1. The confirmation of (7) causes refutation of the assumption (6) (by applying the definition of Rule 1) and transformation of the attribute *Module.author* to an entity type (figure 7). It also leads the generation of a new assumption: (18) $< Sim_Rel \ ; \ Responsible_{Modules} \ ; \ R_{Mod-Comp} >$

2. In order to eliminate the cycle from the subtyping hierarchy one of the two assumptions (16) or (17) must be refuted. If we refute (16), the assumption (14) must, then be refuted, since an entity type can not be its own subtype. In this case the assumption (4) must also be refuted (application of Rule 1).

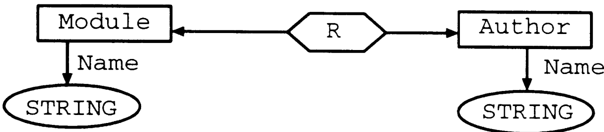

Figure 7

3. In order to eliminate the multiple inheritance from the subtyping hierarchy, we choose to maintain only the most specialized links.

4. The rest of conflicts case are all name conflicts which can be solved by the application of renaming operations. We first begin by confirmingassumptions which have as arguments types with complex structures like assumption (8). The confirmation of this assumption causes the confirmation of (1) and the refutation of { (2), (3),

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

(5), (14) (13) } (application of Rule 1). At this point the only assumption which must be validated by the user is (18). We suppose it will be confirmed.

Figure 8: The integration of "Modules" and "Mod-Comp"

The schema comparison and conforming are now finished. The merging of these schemas produces the schema depicted by (figure 8).

6 Schema quality criteria and their preservation

The use of the data model concepts for designing a schema should result in a schema that obeys some quality criteria. Furthermore, the schema evolution process should result too in a schema that verifies these quality criteria. Hereafter we first present the criteria and then show that the integration process preserves them.

6.1 Schema quality criteria

We consider two kinds of schema quality criteria. The local criteria that are applied to individual types and global criteria which restricts the the successive use of the subtyping relationship.

6.1.1 Local criteria

Entity types belonging to the same schema must have distinct names. The same property must be verified by relationship types and basic types. The main important local criteria concern the use of names. So, entity types, relationship types and basic types of a schema must have distinc names. In addition, attributes, components and operations defined in an entity type or a relationship type must have distinct names.

6.1.2 Global criteria

The global criteria can be viewed as contrainsts defined on the entity types hierarchies (subtyping and composition).

1. **The subtyping hierarchy is a tree**
   This criteria is a consequence of the single inheritance imposed by our data model. To be a tree, the subtyping hierarchy must not contain cycles and every entity type can not have most than one supertype. The checking of the cycle's lack in the subtyping hierarchy avoids misdefinitions of entity types. This criteria can be formalized by :

   $\forall\, e_1 \in Ent \; \forall\, e_2 \in Ent \; \forall\, e_3 \in Ent \quad e_1 <_S^* e_2 \;\wedge\; e_1 <_S^* e_3 \;\Longrightarrow\; e_2 = e_3$
   $\forall\, e_1 \in Ent \; \forall\, e_2 \in Ent \quad e_1 <_S^* e_2 \;\wedge\; e_2 <_S^* e_1 \;\Longrightarrow\; e_1 = e_2$ ($<_S^*$ is acyclic)
   $\forall e \in Ent, \quad < e, e > \notin <_S$ ($<_S$ is acyclic).

2. **The subtyping hierarchy of a schema is a connected_graph**
   This criteria means that all entity types must be related by the subtyping relationship. Moreover, all subtyping hierarchies of a database share a same root called *Entity*. This contrainsts requires the reuse of already defined schema components. This can be formalized by :

   $\forall\, e_1 \in Ent \quad \exists\, e_2 \in Ent \; Such\ that\; e_1 <_S^* e_2 \;\vee\; e_2 <_S^* e_1$
   $\exists\, e,\; e \in Ent \; such\ that\; \forall\, e_1 \in Ent \; e_1 <_S^* e \wedge name(e) = "Entity"$

Proposition 6.1 *The schemas integration process presented in this paper preserves the schema quality criteria.*

proof
Let $Sch_1 =< Ent_1 \,;\, Bas_1 \,;\, Rel_1 \,;\, CONSTR_1 \,;\, SUB_1 \,;\, COMP_1 >$ and $Sch_2 =< Ent_2 \,;\, Bas_2 \,;\, Rel_2 \,;\, CONSTR_2 \,;\, SUB_2 \,;\, COMP_2 >$ be two well defined schemas (a schema is well defined if it verifies the schema quality criteria). We want to prove that the schema $Integ(Sch_1, Sch_2)$ resulting from the integration of Sch_1 and Sch_2 is well defined too.

1. *Names unicity*
   This property is preserved by means of renaming operations which are part of our schemas integration method.

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

2. *Subtyping hierarchy's properties*
Let $SUB_1 = < Ent_1 ; <^1_S>$, $SUB_2 = < Ent_2 ; <^1_S>$ and $SUB = < Ent ; <_S>$. Is SUB a connected_graph and a tree? Note that the fact that SUB share the same root ($Entity$) with Sch_1 and Sch_2 is trivial.

Is SUB a connected_graph?

SUB may not be a connected_graph if and only if $Ent_1 \cap Ent_2 = \emptyset$. Whereas $Ent_1 \cap Ent_2$ is never empty, it contains at least the type $Entity$

Is SUB a tree?

SUB could loose the tree if it contains a cycle or :
$\exists\, e_1 , e_2 \in Ent_1 \quad \exists\, e'_1 , e'_2 \in Ent_2$ *Such that*
$< e_2 , e_1 > \in <^1_S \wedge < e'_2 , e'_1 > \in <^2_S \wedge <$ **Sim_Ent** $; e_2 ; e'_2 >\in LC$
(e_2 *and* e'_2 *are similar*) $\wedge$
$<$ **Sim_Ent** $; e_1 ; e'_1 >\in LI$ (e_1 *and* e'_1 *are different*)
Figure 9 shows such a situation. Let us consider these two last cases and keep the acyclic property for later. In this case the integration of SUB_1 and SUB_2 produces

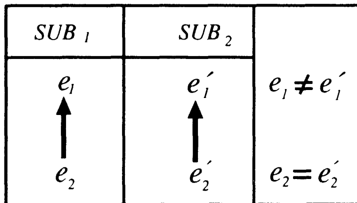

Figure 9

the following result :
$< integ(e_2, e'_2), e_1 >\in <_S \wedge < integ(e_2, e'_2), e'_1 >\in <_S$ with $e_1 \neq e'_1$ (1).
To conserve the tree property, two cases must be considered :

(a) e_1 and e_2 are different but the comparison of Sch_1 and Sch_2 had produced a direct or indirect subtyping link between them. In this case (1) can be replaced by :
$< integ(e_2, e'_2), e_1 >\in <_S$ if $e_1 <^*_S e'_1$, or
$< integ(e_2, e'_2), e'_1 >\in <_S$ if $e'_1 <^*_S e_1$

(b) There is not a subtyping link between e_1 and e'_1. Such a situation is depicted by figure 10. In this case, deleting one of the two subtyping links relating $Integ(e'_4, e_4)$ to its two direct supertypes (e_3 and $Integ(e'_2, e_2)$) causes information loss. Two solutions are proposed to avoid this loss :
Let us suppose that the subtyping link between $Integ(e'_4, e_4)$ and $Integ(e'_2, e_2)$ has been chosen to delete.

- The first solution consists on creating a new subtyping link between $Integ(e'_2, e_2)$ and e_3 (figure 11). New subtyping conflicts can then be produced and their resolution must be made recursively with the aid of the designer (figure 11). This first solution must be done very carefully, since it can introduce "incoherences" in the real world semantic of data.

Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

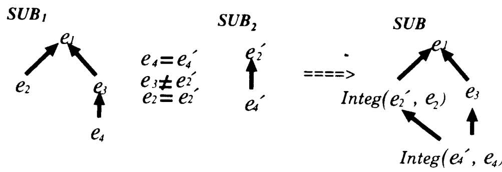

Figure 10

- The second solution consist on the migration of some properties of $integ(e_2, e'_2)$ to the most specialized common supertype of $integ(e_2, e'_2)$ and e_3 which in our case is e_1.

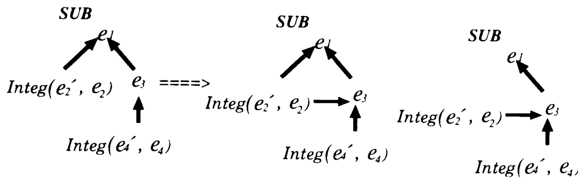

Figure 11

Is SUB acyclic?

SUB should contain a cycle if:
$\exists\ e_1\ ,\ e_2\ \in\ Ent_1\quad Such\ that\ e_1\ <^{*1}_{S}\ e_2\ (1)$
$\exists\ e'_1\ ,\ e'_2\ \in\ Ent_2\quad Such\ that\ e'_1\ <^{*1}_{S}\ e'_2\ (2) \wedge <$ **Sim_ent** $;\ e_1\ ;\ e'_2 > \in\ LC\ \wedge$
$<$ **Sim_ent** $;\ e_2\ ;\ e'_1 > \in\ LI.$
In this situation the schemas merging process produces:
$integ(e_1\ ,\ e'_2)\ <^{*}_{S}\ integ(e_2\ ,\ e'_1) \wedge$
$integ(e'_1\ ,\ e_2)\ <^{*}_{S}\ integ(e'_2\ ,\ e_1) \wedge$
$integ(e'_1\ ,\ e_2)\ \neq\ integ(e'_2\ ,\ e_1)$

This result is a typical case of the cycle's presence in SUB. This case can denote a design mistake which must be fixed by the designer. The integration process consist on checking the presence of such cycles and let the decision of deleting them to the designer. The integration process can not be done without fixing such a conflict in representing data.

7 Conclusion

The process we have presented in this paper allows a safe way to evolve a CASE database schema. Instead of performing schema evolution by means of atomic operations, the use of schema integration avoids some errors and mistakes due to the complexity of CASE databases. We have developed an interactive schema integration process based on assumption management combined with complex schema transformation operations. Data quality criteria are specified by a set of contrainsts which prevent misconceptions of

data schema and restrict the successive use of the abstraction constructs that are offered by the data model. The framework resulting from this study is actually a prototype implemented under the object-oriented system *Smalltalk ObjectWorks*.

Future research consists of analyzing the impact of complex schema transformation on the quality of programs which act on their instances. We are defining a set of programming rewriting rules by using the knowledge given by the schema integration process. These rules must allow programs adaptation and reuse with respect of some program quality criteria.

References

[1] Philip A. Bernstein. Database system support for software engineering– an extended abstract–. In *9th Internat. Conf. on Software Engineering*, pages 166–178, 1987.

[2] S.B. Navathe C. Batini, M. Lenzerini. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surveys.*, 18:323–364, 1986.

[3] P.P.-S Chen. The entity-relationship model: Towards a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, Mars 1976.

[4] E. Codd. A Relational Model of Data for Large Shared DataBases. *Communication of the ACM*, June 1970.

[5] Banerjee J. et al. Semantics and implementation of schema evolution in object-oriented databases. *ACM SIGMOD conference, SIGMOD Record*, 16(3), 1987.

[6] R. Hull and R. King. Semantic database modeling survey applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[7] Paul Johannesson. Schema transformations as an aid in view integration. In *Proc. of CAISE 93*, June 1993.

[8] T. Khammaci and N. Boudjlida. An object-constructor database model to software process modeling. In *Fifth Internat. Sympo. on Computers and Information Sciences*, Cappadocia Turkey, October 1990.

[9] Simon R. Monk and Ian Sommerville. A model for versioning of classes in object-oriented databases. In P.M.D. Gray and R.J. Lucas, editors, *BNCOD 10*, pages 42–58, Aberdeen, 1992. Springer-Verlag.

[10] Maria et al. Orlowska. Schema evolution - the design and integration of fact-based schemata. In *Proc. of the 3rd Australian Database Conference*, 1992.

[11] M. Kracker P. Frankhauser and E. Neuhold. Semantic vs. structural resemblance of classes. *ACM SIGMOD Record*, 4(20):59–63, 1991.

[12] D.J. Penney and J. Stein. Class modification in the gemstone object-oriented dbms. *SIGPLAN Notices (Proc. OOPSLA '87)*, 22(12):111–117, 1987.