

## **A PROPOSAL FOR FORMALIZING AND EXTENDING THE GENERALIZATION AND SUBSET ABSTRACTIONS IN THE ENTITY-RELATIONSHIP MODEL**

Luiz Tucherman, Marco A. Casanova, Pedro M. Gualandi, Anelise P. Braga

Rio Scientific Center - IBM Brasil  
P.O Box 4624  
20.001, Rio de Janeiro, RJ

*Special constructs that formalize and extend the generalization and subset abstractions are first proposed. Then, several operations to maintain sets of objects organized according to the constructs are analysed. Finally, the constructs and operations are given a precise semantics that brings up some key aspects of the entity-relationship model.*

### **1. INTRODUCTION**

Among the abstraction principles used for conceptual design we find the generalization and subset hierarchies. The common definition for the generalization hierarchy is based on the existence of a selection condition (predicate) that divides a class into several subclasses according to the result of the selection. This could even exhaustively partition the class into all possible subclasses.

For the subset hierarchy, the majority of the definitions do not depend on a selection condition and state that the specialization of an entity class  $E$  does not imply the partition of  $E$ . This means that some entity in  $E$  may not be in any of the subclasses, so the only condition imposed by this type of hierarchy is that all entities of a subclass must also occur in  $E$ . Some authors also differentiate the subset hierarchy from the generalization hierarchy based on the possibility of having overlapping or nonoverlapping subsets, respectively.

We first propose in this paper a construct that captures and extends the generalization and subset abstractions. We then suggest several interesting operations to maintain entity and relationship sets organized according to this construct. Finally, we discuss several simple transformations that simplify entity-relationship schemas.

Extensions to the original Entity-Relationship Model [5] can be found in [3,4,6,7,10,11,12]. In particular, proposals to capture the generalization and subset hierarchies can be found in [3,6,8,9,11,12].

The paper is divided as follows. Section 2 describes the syntax and semantics of the constructs. Section 3 introduces operations to manipulate sets organized according to the constructs. Section 4 discusses optimizations and consistency checks for entity-relationship schemas defined using the constructs. Finally,

section 5 contains the conclusions.

## 2. CONSTRUCTS TO CAPTURE GENERALIZATION AND SUBSET HIERARCHIES

### 2.1 Syntax

We let the database designer organize the entity and relationship schemes of a given entity-relationship schema (or, abbreviated, an ER-schema) [13] as a specialization graph with the following characteristics.

First of all, the specialization graph need not be a tree, that is, a strict hierarchy. In fact, we do not even require that the graph be acyclic, although we argue in section 4 that cyclic specialization graphs better be avoided.

An entity scheme  $E$  may be defined as a *qualified* specialization of another entity scheme  $F$ , with qualification  $C$ , which implies that  $E$  will always denote the set of entities of  $F$  that satisfy  $C$ . Hence, we require that an entity scheme  $E$  be a qualified specialization of at most one other entity scheme.

An entity scheme  $E$  may also be defined as a *simple* specialization of  $F$ , indicating that  $E$  must always denote a subset of the set of entities associated with  $F$ . Thus, an entity scheme can be a simple specialization of more than one scheme and, perhaps at the same time, be a qualified specialization of an entity scheme.

We also let a relationship scheme be defined as a simple specialization of another relationship scheme. We rule out qualified specializations in this case because they could create complex interrelationships among entity and relationship sets that would be difficult to enforce. We require that, if  $R$  is a relationship scheme over  $P_1, \dots, P_n$  and  $S$  is a relationship scheme over  $Q_1, \dots, Q_n$  such that  $S$  is a specialization of  $R$ , then  $Q_i$  is either equal to or a specialization of  $P_i$ . This requirement guarantees that it is semantically meaningful to compare  $S$  and  $R$ .

We also introduce the concepts of totality and mutual exclusion for simple specializations. If we declare that  $O$  is *totally specialized* into  $O_1, \dots, O_n$ , then any object in  $O$  must belong to  $O_i$ , for some  $i$ . Orthogonally, if we declare that  $O_1, \dots, O_n$  are *mutually exclusive* then no object in  $O$  may belong to more than one  $O_i$ , for all  $i$ . These are therefore integrity constraints relating  $O, O_1, \dots, O_n$ .

We permit totality and mutual exclusion only for simple specializations for the following reasons. First, since qualified specializations define sets of objects based on attribute values, the database designer can, in this case, express mutual exclusion by using an appropriate definition for the qualifications and totality by defining an assertion for the generalized scheme. However, he cannot apply the same strategy for simple specializations since the sets of objects in this second case are defined "manually" when users insert the objects in the sets.

Finally, we let the ER-schema specify more than one specialization declaration for the same scheme  $O$ , but there cannot be more than one declaration indicating that a scheme  $O$  is a specialization of a scheme  $P$ .

Before proceeding, we introduce some auxiliary definitions. Let  $S$  be an ER-schema. The *specialization graph* of  $S$ ,  $G(S) = (V, A)$ , is a directed graph such that  $V$  is the set of names of the schemes of  $S$  and a pair  $(O, P)$  is in  $A$  iff  $O$  is a simple or a qualified specialization of  $P$  in  $S$ .

Given  $F, G \in V$ , we say that  $G$  is an *transitive specialization* of  $F$  and that  $F$  is an *transitive generalization* of  $G$  iff there is a path from  $G$  to  $F$  in  $G(S)$ ; we say that  $G$  is a *qualified transitive specialization* of  $F$  and that  $F$  is a *qualified transitive generalization* of  $G$  iff there is a path from  $G$  to  $F$  in  $G(S)$  such that, for all arcs  $(H, I)$  of the path,  $H$  is a qualified specialization of  $I$ .

Given  $E, F, G \in V$ , we say that  $E$  is *between*  $G$  and  $F$  iff  $G$  is a transitive specialization of  $E$  and  $E$  is a transitive specialization of  $F$ . Given a set of nodes  $\mathbf{N}$ , we will denote the union of the set of transitive generalizations of the nodes in  $\mathbf{N}$  by  $gen^*(\mathbf{N})$ . Finally, given a node  $G$  and a set  $\mathbf{N}$  of transitive generalizations of  $G$ , the union of the sets of nodes between  $G$  and each node in  $\mathbf{N}$  by *between* $(G, \mathbf{N})$ .

For simplicity, we introduce  $T$  (*top*) as an artificial node such that  $T$  is a transitive generalization of every node and no node is a generalization of  $T$ . Hence, we have that  $gen^*({T}) = \emptyset$  and *between* $(E, {T}) = gen^*(E)$ , for every node  $E$ .

A set of schemes  $\mathbf{N}$  *dominates* a scheme  $E$  iff all immediate generalizations of  $E$  are between  $E$  and  $\mathbf{N}$  (which implies that  $E$  is a transitive specialization of all nodes in  $\mathbf{N}$ ).

We say that an attribute  $A$  (or a key  $K$ ) is *native* to  $O$  iff  $A$  is the name of an attribute (or  $K$  is a key) defined in the scheme whose name is  $O$ . We say that an attribute  $A$  (or a key  $K$ ) is *inherited by*  $O$  from  $P$  iff  $A$  is the name of an attribute of  $P$  (or  $K$  is a key of  $P$ ) and  $O$  is a transitive specialization of  $P$ . Note that we must specify the originating scheme of  $A$  (or  $K$ ) since  $O$  may be a transitive specialization of schemes whose attributes may have the same name (or whose keys may be equal). Thus, in the context of the scheme  $O$  and whenever necessary, we use the notation  $P.A$  (or  $P.K$ ) to refer to the attribute (or key) inherited by  $O$  from  $P$ .

More precisely, we introduce a *specialization declaration* as any expression of the form:

specialize  $O$  [totally] [exclusively]  
into  $O_1$  [where  $C_1$ ], ...,  $O_m$  [where  $C_m$ ]

We say that  $O$  is the scheme *specialized* in the declaration and that  $O_i$  is a *specialization* of  $O$ , which is *simple*, if "where  $C_i$ " is not specified, and *qualified*, otherwise. We also say that  $O$  is a *generalization* of  $O_1, \dots, O_m$ , which is *total*, if "totally" is specified. Likewise, if "exclusively" is specified, we say that  $O_1, \dots, O_m$  are *mutually exclusive* specializations of  $O$ .

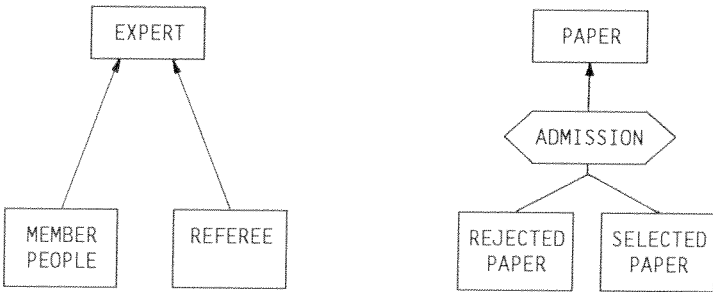
For specializations, we require that:

S0.  $O, O_1, \dots, O_m$  must be distinct names of schemes defined in the ER-schema;

- S1. for each  $i$  in  $[1,m]$ ,  $C_i$ , if specified, must be a qualification over attributes inherited or not, of  $O$ ;
- S2. for each  $i$  in  $[1,m]$ , there cannot exist another specialization declaration indicating that  $O_i$  is a specialization of  $O$ ;
- S3. for each  $i$  in  $[1,m]$ , if  $O_i$  is a qualified specialization of  $O$  then  $O_i$  must not be a qualified specialization of any other entity scheme;
- S4. if 'totality' or 'exclusively' are specified, then  $O_1, \dots, O_m$  must all be simple specializations;
- S5. if  $O$  is the name of a relationship scheme, then no qualification can be specified;
- S6. if  $O$  is the name of a relationship scheme over  $P_1, \dots, P_n$  then, for each  $i$  in  $[1,m]$ ,  $O_i$  must be the name of a relationship scheme over schemes  $Q_1, \dots, Q_n$  such that  $Q_j$  is either equal to or a transitive specialization of  $P_j$ , for each  $j$  in  $[1,n]$ .

In view of S4, it is necessary to specify two separate specialization declarations for a scheme  $O$  when the simple specializations of  $O$  form a total or exclusive classification of  $O$  and  $O$  has some qualified specialization.

To illustrate the use of the construct, consider the following ER diagram representing a fragment of the conceptual schema given in [1]:



In the above ER diagram the generalization hierarchy of the entity PAPER results in two disjoint subsets, REJECTED\_PAPER and SELECTED\_PAPER, which are produced by partitioning PAPER based on the two different values of the attribute ADMISSION. The subset hierarchy of the entity class EXPERT results in two overlapping subsets MEMBER\_PEOPLE and REFEREE, which can be produced by partitioning the generic entity EXPERT by values of different attributes or based on the application.

We can define these two types of hierarchies in different ways, using the specialization declaration, as we do not distinguish the generalization and subset hierarchies.

To define the generalization hierarchy of PAPER, we use the following qualified specialization declaration:

```

specialize PAPER
  into
    REJECTED_PAPER where ADMISSION = 'REJECTED',
    SELECTED_PAPER  where ADMISSION = 'SELECTED';

```

The subset hierarchy of EXPERT can be defined in two ways depending on the existence of some attributes that can be used to classify instances of EXPERT into MEMBER\_PEOPLE and REFEREE. Assuming the existence of attributes MEMBER and REFEREE in EXPERT the following qualified specialization declaration captures the desired hierarchy:

```

specialize EXPERT
  into MEMBER_PEOPLE where MEMBER = 'YES',
    REFEREE          where REFEREE = 'YES';

```

On the other hand, if we do not have those attributes we can define the subset hierarchy of EXPERT through a simple specialization declaration:

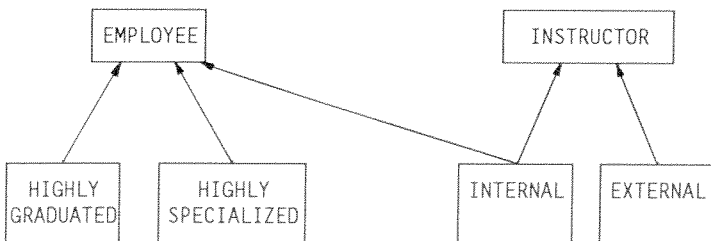
```

specialize EXPERT
  into MEMBER_PEOPLE, REFEREE;

```

Note that neither the option `totally` nor the option `exclusively` has been specified because, for the subset hierarchy, as defined in the literature, we can have instances of a generic class that do not belong to any subclass and, moreover, we can have overlapping subclasses. In the previous example, we can have instances of EXPERT that belong to both MEMBER\_PEOPLE and REFEREE subclasses, to one of them and to neither of them.

The next example illustrates how to combine the qualified and simple specialization hierarchies to capture very complex concepts from the real world. Consider the following fragment of an ER-diagram:



In this example, we classify EMPLOYEE into two qualified specializations, HIGHLY\_GRADUATED and HIGHLY\_SPECIALIZED, and into one simple specialization, INTERNAL, and we categorize INSTRUCTOR into two qualified specializations, INTERNAL and EXTERNAL, that are disjoint subsets. Thus INTERNAL represents an

employee that teaches some course. The appropriate specialization declarations are

```
specialize INSTRUCTOR
  into EXTERNAL where TYPE = 'EXTERNAL',
       INTERNAL  where TYPE = 'INTERNAL';

specialize EMPLOYEE
  into INTERNAL,
       HIGHLY_GRADUATED where EDUCATION = 'PHD' or
                               EDUCATION = 'MSC',
       HIGHLY_SPECIALIZED where SPECIALIZATION = 'TECHNICAL' and
                               EXPERIENCE >= 10;
```

Note that we can have instances of EMPLOYEE that are not instances of the two qualified specializations, but which are instances of INTERNAL. Moreover, we can also have instances of EMPLOYEE that cannot be classified in any of the three subclasses.

## 2.2 Semantics

This section is concerned with the notions of state and consistent state of an ER-schema. Briefly, a state of an ER-schema  $S$  maps each entity scheme of  $S$  into a set of entities, taken from a common set  $E$ , each relationship scheme into a subset of the cartesian product of the sets of entities involved, and each attribute of a scheme into a function that maps each object into a value taken from the appropriate domain. It is very important to stress that, unlike the relational model, entities and relationships have an existence independent of their attribute values, which are necessary however to locate them in the database [2].

We use  $E^n$  to denote the  $n$ -fold cartesian product of  $E$  with itself and recall that  $E^m \times E^n$  is isomorphic to  $E^{m+n}$ . Hence, we will treat an element of  $E^m \times E^n$  as belonging to  $E^{m+n}$  and vice-versa.

We factor out the semantics of the domain types by introducing the concept of a *domain function*  $D$  as any function that maps each valid domain type  $D$  into a set  $D(D)$ .

A *state* for an ER-schema  $S$  is a triple  $\alpha = (D, E, I)$ , where:

- 1)  $D$  is a domain function;
- 2)  $E$  is a non-empty set, called the *entity domain*;
- 3)  $I$  is a function satisfying the following conditions:
  - a) the domain of  $I$  is the set of names of all schemes defined in  $S$ , plus the set of all pairs  $(O, A)$  where  $O$  is the name of a scheme and  $A$  is the name of an attribute of  $O$ ;
  - b) if  $E$  is the name of an entity scheme of  $S$ , then  $I(E)$ , the *entity set* of  $E$  in  $\alpha$ , is a finite subset of  $E$ ;
  - c) if  $R$  is the name of a relationship scheme of  $S$  over  $O_1, \dots, O_n$ , then  $I(R)$ , the *relationship set* of  $R$  in  $\alpha$ , is a finite subset of  $I(O_1) \times \dots \times I(O_n)$ ;
  - d) if  $A$  is the name of an attribute of scheme  $O$  of  $S$  and if  $D$  is the domain type

of  $A$ , then  $I((0,A))$ , the *attribute function* of  $A$  of  $0$  in  $\alpha$ , is a function from  $I(0)$  into  $D(D)$ .

Since no two schemes in  $S$  have the same name as well as no two attributes of a scheme have the same name, the domain of the function  $I$  will have exactly one element for each object of the ER-schema  $S$ .

If  $R$  is the name of an  $n$ -ary relationship scheme and  $N$  is the name of the  $i^{\text{th}}$  role of  $R$ , let  $\pi(i,I(R))$  indicate the  $i^{\text{th}}$  projection of a relationship set  $I(R)$  and, likewise, let  $\pi(i,r)$  and  $\pi(N,r)$  indicate the  $i^{\text{th}}$  element of a tuple  $r$  in  $I(R)$ .

A state  $\alpha=(D,E,I)$  of  $S$  is *consistent* with respect to a set of specialization declarations iff

- 1) for any two schemes  $0$  and  $P$  of  $S$ , if  $P$  is a simple specialization of  $0$  then  $I(P) \subseteq I(0)$ ;
- 2) for any two entity schemes  $E$  and  $F$  of  $S$ , if  $F$  is a qualified specialization of  $E$ , with qualification  $C$ , then  $I(F) = \{e \in I(E) \mid e \text{ satisfies } C\}$ ;
- 3) for any schemes  $0, 0_1, \dots, 0_m$ , if  $0$  is a total generalization of  $0_1, \dots, 0_m$ , then  $I(0) \subseteq I(0_1) \cup \dots \cup I(0_m)$ ;
- 4) for any two schemes  $0$  and  $P$ , if  $0$  and  $P$  are mutually exclusive then  $I(0) \cap I(P) = \emptyset$ .

From the definition of consistent state, it should be clear that, if  $0$  transitively specializes  $0'$ , then the set of objects associated with  $0$  is always a subset of the set of objects associated with  $0'$ . Therefore, any attribute function associated with an attribute of  $0'$  is defined on all objects in the set associated with  $0$ . This property permits us to extend a consistent state  $\alpha=(D,E,I)$  of an ER-schema  $S$  to give semantics to the inherited attributes as follows:

- 1) the domain of  $I$  is extended to include all pairs of the form  $(0,(0',A))$  where  $A$  is inherited by  $0$  from  $0'$ ;
- 2)  $I((0,(0',A)))$  is the function  $I((0',A))$  restricted to the set  $I(0)$ .

### 3. MAINTAINING SPECIALIZATION GRAPHS

We propose in this section a set of operations to maintain sets of objects organized as a specialization graph. For the sake of simplicity, we only consider specialization graphs of entity sets and keep the discussion at an exploratory and informal level.

We define the operations so that they preserve the semantic constraints pertaining to specializations and respect the concept that each entity exists independently. In particular, the operations will guarantee that the insertion of a new entity into a set always propagates to the transitive generalizations and to the qualified transitive specializations of the set, depending on the qualifications, and that the deletion of an entity from a set always propagates to the transitive specializations and to the qualified transitive generalizations of the set, also depending on the qualifications.

Consider first a *deletion operation* of the form

delete from E where Q

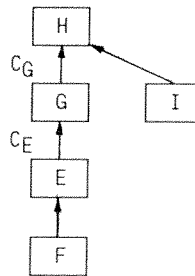
where E is the name of an entity scheme and Q is a selection, whose exact syntax is left unspecified, that defines a subset of the set of entities associated with E in each state of the ER-schema. The semantics of this operation can be briefly described as follows:

- 1) construct the set **D** of all entities in the current set associated with E that satisfy Q (if Q is omitted, take **D** as the set of all entities in the current set associated with E);
- 2) execute  $\text{delete}^*(E, \mathbf{D})$ .

The operation  $\text{delete}^*(E, \mathbf{D})$  is not accessible to the users, that is, it just helps define the semantics of the original delete operation. It is recursively defined as follows:

- 1) if the deletion of all entities in **D** from the set currently associated with E does not violate any totality restriction involving E  
     then remove all entities in **D** from the set currently associated with E;  
     else reject the operation;
- 2) for each F such that F specializes E, execute  $\text{delete}^*(F, \mathbf{D})$ ;
- 3) for each G such that E is a qualified specialization of G do:  
     execute  $\text{delete}^*(G, \mathbf{D})$ ;

For example, consider a specialization graph of the following form:



The deletion of a set of entities from E will first propagate downward to F, then upward to G, via the qualification  $C_E$ , then upward again to H, via the qualification  $C_G$ , and finally downward to I.

We now consider a first form of insertion, that we call *classification*, that takes an entity  $e$  in the sets associated with schemes  $H_1, \dots, H_n$  and inserts  $e$  into a common transitive specialization E of  $H_1, \dots, H_n$ , setting the values of the new attributes of  $e$ . If E is a specialization of some scheme I which is neither between  $H_i$  and E nor a transitive generalization of  $H_i$ , for some  $i \in [1, n]$ , then  $e$  must already be in the set associated with I. If G is between E and  $H_i$ , for some  $i \in [1, n]$ , then  $e$  is indeed inserted into G as a side-effect, if not already present. Note that this is possible since all attributes of G are indeed (inherited) attributes of E and, hence, they must be specified in the classification.

The general format of the *classification operation* is:

classify from  $H_1$  where  $Q_1$  , ... , from  $H_n$  where  $Q_n$   
into E set  $X = \bar{x}$

where, for each  $i \in [1, n]$ ,  $H_i$  is an entity scheme,  $Q_i$  is a selection that defines a subset of the set of entities currently associated with  $H_i$ , E is a transitive specialization of  $H_1, \dots, H_n$ , X is a list of all the attributes native to E or inherited from schemes between E and  $H_1, \dots, H_n$  and  $\bar{x}$  is a list of values for the attributes in X.

The semantics of this operation is:

- 1) for each  $i \in [1, n]$ , let  $I_i$  be the set of all entities, in the current set associated with  $H_i$ , that satisfy  $Q_i$ ;
- 2) let  $I$  be the intersection of  $I_1, \dots, I_n$ ;  
if  $I$  does not contain just one entity,  
then reject the operation;  
else let  $e$  be the only entity in  $I$ ;
- 3) execute  $\text{classify}^*(e, \mathbf{H}, E, X, \bar{x}, \lambda, \lambda)$ , where  $\mathbf{H} = \{H_1, \dots, H_n\}$ .

Consider now the (internal) operation  $\text{classify}^*(e, \mathbf{H}, E, X, \bar{x}, GP, SP)$ , where  $\mathbf{H}$ , E, X,  $\bar{x}$  are as for the classification operation,  $e$  is an entity in the sets associated with the schemes in  $\mathbf{H} \cup \text{gen}^*(\mathbf{H})$  and the last two arguments are used just to avoid redundant work in the recursive calls. The semantics of  $\text{classify}^*$  goes as follows ( $\lambda$  denotes an undefined value):

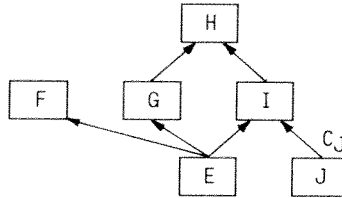
- 1) for each G such that E is a specialization of G and  $G \neq GP$  do
    - a) if  $G \notin \mathbf{H} \cup \text{between}(E, \mathbf{H}) \cup \text{gen}^*(\mathbf{H})$  but  $e$  is not in the set currently associated with G,  
then reject the operation;
    - b) if  $G \in \text{between}(E, \mathbf{H})$  then  
if  $e$  exists in the set currently associated with G,  
then  
if the values of the native attributes of G for  $e$  do not agree with those in  $\bar{x}$ ,  
then reject the operation;  
else do nothing;
  - else  
execute  $\text{classify}^*(e, \mathbf{G}, G, Y, \bar{y}, \lambda, E)$ , where  $\mathbf{G}$  is the intersection of  $\mathbf{H}$  and  $\text{gen}^*(G)$ , Y is the restriction of X to the attributes of G and  $\bar{y}$  is the restriction of  $\bar{x}$  to the attributes in Y;
- 2) for the entity  $e$ , let the values of the native attributes and the attributes inherited from schemes between E and  $\mathbf{H}$  be given by  $\bar{x}$  and let the values of all other attributes of  $e$  be those that  $e$  already has (the previous step assures this);
    - a) if any of the following tests fails, reject the operation:
      - i) if E is a qualified specialization of some entity scheme G with qualification  $C_G$ , test if  $e$  satisfies  $C_G$ ;
      - ii) test if the addition of  $e$  to E will violate any condition imposed by the totality or mutual exclusion keywords of a specialization declaration involving E;
    - b) add  $e$  to the set currently associated with E, with the values of the native

attributes of E given as in the list  $\bar{x}$ .

- 3) for each F such that F is a qualified specialization of E with qualification  $C_F$  and  $F \neq SP$  do  
 if  $e$  satisfies  $C_F$   
 then  
 execute  $\text{classify}^*(e, \mathbf{H}, F, Z, \bar{z}, E, \lambda)$ , where Z is a list of the native attributes of F concatenated with X and  $\bar{z}$  agrees with  $\bar{x}$  on the attributes in X and  $\bar{z}$  is equal to null for the native attributes of F.

Note: the tests in (2a) must be expanded to cover other restrictions imposed by the schema (and not discussed in this paper).

For example, consider the following specialization graph:



Suppose that  $e$  is in the set currently associated with H, but  $e$  is not in G, I or E. Then, to classify  $e$  into E, the user must provide values for all attributes native to E and all attributes inherited from G and I. The classification will succeed if, among other facts,  $e$  already belongs to the set currently associated with F. Note that  $e$  will be inserted into G and I. Moreover, if  $e$  satisfies  $C_J$ ,  $e$  will also be inserted into J; the values of the attributes native to J will be set to null, if the domain type permits, otherwise the insertion of  $e$  into J is rejected and the whole classification fails.

Consider now an *insertion operation* of the form:

`insert into E with X= $\bar{x}$`

where E is the name of an entity scheme, X is a list of all attributes, inherited or not, of E and  $\bar{x}$  is a list of values for the attributes in X. The semantics we give to this statement forces the insertion of a new entity into E, into all transitive generalizations of E and into all qualified transitive specializations of E, if necessary. Note, however, that  $\bar{x}$  will not provide values for the attributes native to the qualified transitive specializations of E, which are then set to null, if possible. Note also that the user who issues such operation must know exactly how to compute the inherited attributes of E and he must be aware of all propagations.

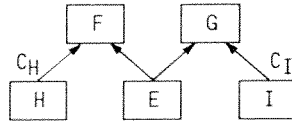
More precisely, the semantics of the operation, in the context of a given specialization graph, is:

- 1) create a new entity  $e$ ;
- 2) execute  $\text{classify}^*(e, \{T\}, E, X, \bar{x}, \lambda, \lambda)$ ;

We emphasize that the first step creates a new entity, that is, an entity not in the

current state, an action that is fully in line with the semantic concept that entities have an independent existence. The second step then inserts the new entity into E and all transitive generalizations of E since, by definition of the "top" node T,  $between(E, \{T\}) = gen^*(E)$ .

For example, consider the following specialization graph:



Suppose that all native attributes of H and I admit null values. Then, to insert a new entity into E, the user specifies values for the native attributes of E and to attributes of E inherited from F and G. The execution of the operation will first create a new entity say,  $e$ , and insert it into E. Then,  $e$  will be inserted into F and G. If  $e$  satisfies  $C_H$ , then  $e$  will also be inserted into H with all native attributes of H set to null and, if  $e$  satisfies  $C_I$ , then  $e$  will likewise be inserted into I.

We now briefly consider another interesting form of insertion. By way of motivation, suppose that E specializes both F and G and that there are entities  $f$  and  $g$  in the sets currently associated with F and G, respectively. Suppose that (in the real world) the user discovers that  $f$  and  $g$  are the same entity and that he wants to classify it into E. We then provide an operation that identifies  $f$  and  $g$  and inserts the identified entity into E. Note that he cannot use the classify operation in this case because (in the database)  $f$  and  $g$  are distinct entities.

The general syntax of the *identification operation* is:

identify from  $H_1$  where  $Q_1$ , ..., from  $H_n$  where  $Q_n$   
 [into E set  $X = \bar{x}$ ]

where, for each  $i \in [1, n]$ , with  $n > 1$ ,  $H_i$  is an entity scheme,  $Q_i$  is a selection that defines a subset of the set of entities currently associated with  $H_i$ , and, if specified, E is a transitive specialization of  $H_1, \dots, H_n$ , X is a list of all the attributes native to E or inherited from schemes between E and  $H_1, \dots, H_n$  and  $\bar{x}$  is a list of values for the attributes in X.

The semantics of this operation is:

- 1) for each  $i \in [1, n]$ , let  $I_i$  be the set of all entities, in the current set associated with  $H_i$ , that satisfy  $Q_i$ . If  $I_i$  does not contain exactly one element, then reject the operation;
- 2) create a new entity  $e$ ;
- 3) for each  $i \in [1, n]$ , replace the only entity  $e_i$  in  $I_i$  by  $e$  in all entity sets, all relationship sets and all attribute functions of the current state where  $e_i$  occurs;
- 4) execute  $classify^*(e, \{H_1, \dots, H_n\}, E, X, \bar{x}, \lambda, \lambda)$

Note: step (3) is executed only if E has been specified.

The reader should at this point carefully analyse the difference between the operations of classification, insertion and identification.

Lastly, we observe that we could also define other useful operations in addition to those described above. For example, we could define operations to move or copy an entity from one set to another or define a partial inverse of the identifier operation.

#### 4. FURTHER RESTRICTIONS AND OPTIMIZATIONS FOR SPECIALIZATIONS

##### 4.1 Further Restrictions

Informally, we say that an ER-schema is *redundant* iff the schema has statements that are partially implied by others and we say that the schema is *ill-defined* iff all consistent states of  $S$  must assign the empty set to some scheme of  $S$ .

Let  $S$  be an ER-schema and let  $G(S) = (V, A)$  be the specialization graph of  $S$ . We can first prove that:

**Proposition 1:** Let  $\alpha = (D, E, I)$  be a consistent state of  $S$ . If there is a path from  $O$  to  $P$  in  $G(S)$  then  $I(O) \subseteq I(P)$ .

Observe that all schemes participating in a cycle  $(O_0, \dots, O_n, O_0)$  of  $G(S)$  will always denote the same set of objects in any consistent state since, by the proposition, we would have  $I(O_0) \subseteq \dots \subseteq I(O_n) \subseteq I(O_0)$ . Therefore, we can optimize  $S$  by collapsing  $O_0, \dots, O_n$  into a single scheme  $O$  that has all the attributes originally defined for each of the collapsed schemes and modifying the definition of the other statements of the ER-schema to reflect such transformation. However, this course of action has deep consequences for the structure of the ER-schema and for the semantics of the operations. Therefore, we abandon it and require that:

G1.  $G(S)$  must be acyclic.

Now, if  $S$  specifies that  $O$  and  $P$  are mutually exclusive and there is a path in  $G(S)$  from  $O$  to  $P$  then  $S$  is ill-defined since  $I(O)$  must be empty, for any consistent state  $I$  of  $S$ . Likewise, if  $O$  and  $P$  are mutually exclusive and they have a common transitive specialization  $Q$ , then  $S$  is ill-defined since  $I(Q)$  must be empty, for any consistent state  $I$  of  $S$ . Therefore, we further require that:

G2. if  $S$  specifies that  $O$  and  $P$  are mutually exclusive then there must be no path in  $G(S)$  from  $O$  to  $P$ .

G3. if  $S$  specifies that  $O$  and  $P$  are mutually exclusive then there must be no common transitive specialization of  $O$  and  $P$ .

All conditions discussed above are easily detected because they reduce to finding paths in a graph. We now discuss additional criteria that depend on an analysis of the qualified specializations declared in an ER-schema as well as the static assertions possibly defined for entity or relationship schemes.

Let  $\alpha = (D, E, I)$  be a consistent state of an ER-schema  $S$ . Let  $O$  be a scheme of  $S$  and let  $o$  be an object in  $I(O)$ . Then,  $o$  satisfies all static assertions define for the scheme  $O$ . Moreover, if  $O$  is a simple specialization of  $P$ ,  $o$  must also belong to  $I(P)$ , which implies that  $o$  also satisfies all static assertions associated with  $P$ . However, if  $O$  is a qualified specialization of  $P$  with qualification  $C$ , then  $o$  satisfies all static assertions associated with  $P$  and also the qualification  $C$ . All these observations can be captured as follows.

Let  $S$  be an ER-schema and assume that  $G(S)$  is acyclic. For each scheme  $O$  of  $S$ , define  $\overline{Q}(O)$  as follows:

$$\begin{aligned} \overline{Q}(O) = C, & \quad \text{if there is a scheme } P \text{ such that } O \text{ is a qualified specialization} \\ & \quad \text{of } P \text{ with qualification } C \\ \overline{Q}(O) = \text{true}, & \quad \text{otherwise} \end{aligned}$$

Since a scheme can be a qualified specialization of at most one other scheme,  $\overline{Q}$  is well defined. Also define  $\overline{A}(O)$  as the conjunction of all static assertions defined for  $O$ .

The *augmented specialization graph* of  $S$ ,  $H(S) = (V, A, m)$ , is a directed graph with labelled nodes such that  $V$  is the set of names of the schemes of  $S$  and a pair  $(O, P)$  is in  $A$  iff  $O$  is a simple or qualified specialization of  $P$  in  $S$ . The node labelling function  $m$  is defined as follows:

- 1) If  $O$  has out-degree 0 then  $m(O) = \overline{A}(O)$ ;
- 2) If  $O$  has out-degree greater than 0 and if the arcs leaving  $O$  are  $(O, O_1), \dots, (O, O_n)$  (that is, if  $O$  specializes  $O_1, \dots, O_n$ ) then  $m(O) = (m(O_1) \wedge \dots \wedge m(O_n)) \wedge \overline{Q}(O) \wedge \overline{A}(O)$

Note that  $m$  is well-defined because we assumed that  $G(S)$ , and hence  $H(S)$ , are acyclic. Moreover,  $m(O)$  is defined only over attributes of  $O$  (and not over attributes of transitive specializations of  $O$ ). We can then prove that:

**Proposition 2:** Let  $\alpha = (D, E, I)$  be a consistent state of  $S$ . Then every  $o \in I(O)$  satisfies  $m(O)$ .

Our final requirement is:

G4. for no scheme  $O$  defined in  $S$ ,  $m(O)$  is false.

Note that this last condition is much more difficult to enforce since it requires a theorem prover to cope with the formulas expressing qualifications and static assertions.

Finally, we can show that taking totality into account will not help detect new inconsistencies.

#### 4.2 Simple Optimizations

Using the specialization graph, we obtain three simple optimizations:

### Optimization of Simple Specializations:

Suppose that  $(0,P)$  is an arc in  $G(S)$  corresponding to a simple specialization and that there is a path  $(0,Q,\dots,P)$  in  $G(S)$ . Then, replace  $S$  by a new ER-schema  $S'$  obtained by dropping from  $S$  the specification that  $0$  is a simple specialization of  $P$ .

The new ER-schema  $S'$  is indeed equivalent to  $S$  because the alternate path from  $0$  to  $P$  already determines that  $I(0)$  must be a subset of  $I(P)$  in any consistent state  $I$  hence making the simple specialization redundant.

### Optimization of Qualified Specializations (1):

Let  $(0,P)$  be an arc of  $G(S)$  corresponding to a qualified specialization of  $S$  with qualification  $C$ . Suppose there is a path in  $S$  of the form  $(0,Q,\dots,P)$ . Then, replace  $S$  by a new ER-schema  $S'$  obtained by:

- 1) dropping from  $S$  the specification that  $0$  is a qualified specialization of  $P$ ;
- 2) defining that  $0$  is now a qualified specialization of  $Q$  with qualification  $C'$ , obtained by changing each occurrence of an attribute  $A$  in  $C$  by  $P.A$ , to reflect that  $P.A$  is now an attribute of  $Q$  inherited from  $P$ .

The syntactic correctness of the new ER-schema  $S'$  follows first because  $Q$  inherits all the attributes of  $P$ , which implies that  $C$  can indeed be redefined over  $Q$ . Second because, by the restrictions of the language,  $0$  cannot be a qualified specialization of  $Q$  in  $S$  since  $0$  is already a qualified specialization of  $P$ . Hence, step 2 can indeed be carried out without violating the restrictions of the language. Finally,  $S'$  is equivalent to  $S$  because, in any consistent state  $I$  of  $S$ , the qualified specialization requires that  $I(0)$  be the set of objects of  $I(P)$  that satisfy  $C$ ; but the alternate path from  $0$  to  $P$  determines that  $I(0) \subseteq I(Q) \subseteq I(P)$ ; hence,  $I(0)$  is the set of objects in  $I(Q)$  that satisfy  $C$ .

Let  $S$  be an ER-schema and  $H(S) = (V, A, m)$  be the augmented specialization graph of  $S$ . Using  $H(S)$ , we obtain an additional optimization whose correctness follows from Proposition 2:

### Optimization of Qualified Specializations (2):

If  $0$  is a qualified specialization of  $P$ , with qualification  $C$ , and if  $m(P)$  implies  $C$ , then replace  $S$  by a new ER-schema  $S'$  obtained by collapsing  $0$  into  $P$ .

## 5. CONCLUSIONS

The constructs presented in section 2 formalize and extend the generalization and subset abstractions within the context of the entity-relationship model, while the operations in section 3 offer interesting alternatives for developing a data manipulation language that takes these abstractions into account.

Any implementor of an entity-relationship database management system should, in a first stage, carefully analyse the semantics of the constructs and the operations to assess their complexity. On a second stage, the implementor would perhaps want

to further restrict the constructors and operations to reduce the implementation cost to the desired level.

## REFERENCES

- [1] P. Bertaina, A. Di Leva, P. Giolito, Logical design in Codasyl and relational environment, in S. Ceri (ed.), *Methodology and Tools for Data Base Design*, North-Holland, 1983.
- [2] A. Borgida, "Features of languages for the development of Information Systems at the Conceptual level", *IEEE Software* 2(1), January 1985, 63-73.
- [3] U. Bussolati, S. Ceri, V. De Antonellis and B. Zonta, Views Conceptual Design, in S. Ceri (ed.) *Methodology and Tools for Data Base Design*, North-Holland, 1983.
- [4] S. Ceri (ed.), *Methodology and tools for data base design*, North-Holland, 1983.
- [5] P. P. Chen - The entity-relationship model: toward a unified view of data - *ACM TODS*, 1, 1 (1976) 9-36.
- [6] A. Dogac, P.P. Chen, Entity-Relationship Model in the ANSI/SPARC Framework, in *Entity-Relationship Approach to System Analysis and Design*, P.P. Chen (ed.), North-Holland (1981), 361-378.
- [7] R. Elmasri, J. Weeldreyer, A. Heuner, The Category Concept: An extension to the entity-relationship model, in *Data and Knowledge Engineering I*, North-Holland, 1985, 75-116.
- [8] H.F. Korth and A. Silberschatz, *Database System Concepts*, McGraw-Hill Book Co. (1986).
- [9] W. Kozaczynski, L. Lillien, An Extended Entity-Relationship (E<sup>2</sup>R) Database Specification and its Automatic Verification and Translation into the Logical Relational Design, *Proc. of the Sixth Int. Conf. on Entity-Relationship Approach*, New York (Nov. 1987), 497-513.
- [10] M. Lenzerini, G. Santucci, Cardinality constraints in the E-R model, *Entity-Relationship Approach to Software Engineering*, North-Holland, 1983.
- [11] P. Scheuermann, G. Schiffner, H. Weber, Abstraction capabilities and invariant properties modelling within the Entity-Relationship approach, P.P. Chen (ed.) *Entity-Relationship approach to System Analysis and Design*, North-Holland, 1980.
- [12] T.J. Teorey, D. Yang, J.P. Fry, A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, *ACM Computing Survey*, 18, 2 (June 1986), 197-222.
- [13] L. Tucherman, M.A. Casanova, P.M. Gualandi, A.P. Braga and M.R. Cavalcanti, "A Data Definition Language for an Extended Entity-Relationship Model", Technical Report CCR072, Rio Scientific Center, IBM Brazil (June 1989).