FACULTÉS UNIVERSITAIRES
NOTRE-DAME DE LA PAIX
NAMUR, BELGIUM

Institut d'Informatique

Rue Grandgagnage, 21
B-5000 Namur
BELGIUM

A Generic Entity-Relationship Model

Jean-Luc Hainaut

RP-89-001 January 1989

Phone: +32 81 72.49.66 Fax: +32 81 72.49.67 E-mail: cleroy@info.fundp.ac.be

A GENERIC ENTITY-RELATIONSHIP MODEL

J-L Hainaut

Institut d'Informatique
University of Namur, Belgium

The objective of the paper is to put forward a general framework allowing the specification, comparison and conversion of most information and data models currently proposed for Information System design, be they at the conceptual or at the DBMS levels. That framework is based on an extended relational model that includes the concepts of entity domain and of complex domain. The analysis of the constructs available in current information models, and particularly E-R models, demonstrates the ability of the extended relational model to express these constructs without loss of semantics. The *Generic Entity-relationship (GER) model* is a large subset of the extended relational model that can uniquely represent the concepts of most current information models, while being more regular, less complex and more powerful than their union. By further augmenting the GER with two simple logical implementation constructs we extend it to the expression of DBMS-dependent models.

KEYWORDS : information modeling, Entity-Relationship model, model equivalence, schema equivalence

1. INTRODUCTION

Since the mid-sixties [CODASYL,62],[BACHMAN,69], several hundreds of specification models for the description of complex data structures were published. Most of these proposals tackle the problem of building a machine-independent (or conceptual) description of data bases. Historically speaking, two major trends can be observed, namely the relational model family, which originated in 70 and the Entity-relationship model family popularized since 1976. The evolution of these families have not been independent, and can be considered as convergent at the present time.

The relational model proposes a simple and uniform view of data based on the theory of relations. That theory, based on the agregation of values representing properties of world entities, formalizes both the semantic structures of data (e.g. through the dependency theory) and data designation and derivation (e.g. the relational calculus and algebra) [CODD,70]. This oversimplistic model has been quickly enriched in two ways. The first extension was the introduction of specific constructs for the explicit representation of entities and of their taxonomic relations without making use of identifying properties [HALL,76] [CODD,79]. The second extension was the relaxation of the 1st-normal form hypothesis, leading to compound and multivalued attributes. That extension has led to the so-called Non First Normal Form (N1NF), Nested Relation, Complex Object and

Structured Object models, to mention only some of them [MAKINOUCHI,77] [JAESCHKE,82] [ABITEBOUL,87] [HULL,87]. That increased semantic power has been paid by much higher complexity in both the data view and languages. Some authors even propose to introduce algorithmic constructs in relational algebra for simplification purposes [GYSSENS,88].

The Entity-relationship model born in the early seventies was popularized by [CHEN,76]. At first deeply rooted in the standard relational theory (P. Chen scrupulously gives a relational translation of each E-R construct), it escaped slowly from that context to become an independent model. Several hundreds of papers and books have been published (e.g. in the dedicated *Entity-Relationship* conferences) to propose more precise definitions and extensions for the original E-R model. Let us only mention [PARENT,86] and [KOZACZYNSKI,87]. The very purpose of the E-R models (modeling more real world aspects), its richness (and therefore its complexity) and its attractive graphical representation have led to a rather intuitive description of the model and have delayed its formal definition (be it said in passing, that evolution from practice to formalization is closer to the traditional DBMS history than to the relational history). For instance, the concepts of *model*, *diagram*, and *methodology* are still often confused in the E-R world. By now, some attempts to formalize the model and its languages have appeared [PARENT,86]. These efforts will lead to better understanding of the E-R description of data and of its manipulation. However, they are impaired by the complexity of the task. As an example, consider that SEQUEL (the grandfather of SQL) was designed 3 years after the publication of the relational model, while a widely accepted *ER-SQL* has yet to emerge 14 years after Chen's paper. As a matter of fact, a strong convergence can be observed between N1NF (and relatives) models and recently formalized E-R models as far as structural aspects and algebra/calculus-based languages are concerned.

These major families do not account for other interesting models that have developed concurrently. Let's only mention binary models [ABRIAL,74] [HAINAUT,74] [BRACCHI,76] [VERHEIJEN,82] and functional models [SHIPMAN,81] (a survey of some popular semantic models can be found in [PECKHAM,88]). However, as this paper will try to justify it, these models can be considered as variants of a more general information model.

In all these modern information models, the predominance of the **object-association** philosophy, in which the world is perceived as a collection of objects in association with each other is obvious. They all seem to be based on a very small set of concepts. They mainly differ as to the names of these concepts and as to the way they can be assembled and structured. The purpose of this paper is to build a (fairly) minimal set of concepts and structures that encompass those of these models. These concepts and structures will be designed to be a framework in which each aforementioned model can be described as a subset. Therefore, that framework, called *Generic Entity-Relationship* (GER) model, will allow the specification, comparison and conversion of these models, together with the translation of schemata expressed in different models. The GER is based on two main observations : (1) despite their inherent and historical problems, the Entity-relationship (E-R) models represent the most popular semantic modeling tools for now, and many other approaches can be considered as variants of them, (2) the relational model is a simple, sound and completely formalized framework for information modeling, and the current extensions make it quite able to modelize complex semantics. From those starting ideas, the GER model will be built following a methodology which is reflected in the structure of the paper. In section 2, we shall define an *extended relational model*, based on two extensions, namely an *entity domain* for representing world entities explicitly, and *complex domain constructors* that define powerset or cartesian product domains (among others). In section 3, the concepts of the traditional E-R models are analysed and translated in the extended relational model. The same exercise is practiced on the current semantic

extensions of the E-R models in section 4. The conclusion of that analysis is that it is possible to define a subset of the extended relational model in such a way that a *one-to-one mapping can be stated between that subset and all the constructs of the E-R models* studied in section 3 and 4. Section 5 is devoted to the definition of that subset, which is called the GER model. In section 6, two simple concepts are added to the GER model in order to make it able to capture the main aspects of the current DBMS models, in such a way that both conceptual and implementation aspects of information design and specification can be addressed by the GER model. Section 7 evokes the need for completing the GER model with a set of transformations that guarantee the conversion of models and of schemata without loss of semantics.

The paper will focus only on static, structural aspects of information modeling, leaving aside complementary aspects such as query and manipulation languages or behavioural extension (e.g. abstract data type, object-oriented specification) . It is not concerned with the various diagrammatic representation of E-R schemata, nor with E-R based design methodologies.

The reader is supposed to be acquainted with traditional and current relational theory (see for instance [MAIER,83] [ULLMAN,88]).

2. AN EXTENDED RELATIONAL MODEL

The basic relational model, as stated in [CODD,70] and described in general literature (such as [ULLMAN,88] and [MAIER,83]), relies on the notions of simple value domain, 1NF relation, attribute, relational algebra and dependencies (key, FD, MVD, JD). Since then, many extensions have been proposed in order to augment the power of the relational model to represent more accurately and more naturally the semantic structures of the so-called *Universe of Discourse* (UoD). Two extensions have been of particular importance, namely the *entity domain* and the *non simple domain*. The entity domain is a predefined set of values dedicated to the explicit representation of entities [CODD,79]. A non simple domain is made up of values that are either sets of values or tuples of a relation. Non simple domains are the very basis of *non first normal form*, *nested relation*, *complex* or *structured object* models [HULL,87]. The extended relation model consists of the basic relational model augmented with variants of the above-mentioned extensions.

The definition of the extended model will be illustrated with some examples written in the definition language the syntax of which is outlined in appendix A.

Domains, basic domains and subdomains
A domain is any declared set of similar elements. Some domains are *static*, while others are *dynamic*. In the latter case, the *set* of values can evolve, new elements being added, and others being withdrawn. Among the static domains we can mention named predefined basic domains such as *integer*, *real*, *date* or *char*. The basic *entity domain*, which is given the name ENTITIES, is an arbitrary set of concrete or abstract elements that can be used to denote entities of the UoD [HALL,76],[CODD,79]. Since entities enter the UoD, while others disappear, the ENTITIES domain is dynamic. A subdomain is a named domain defined as a subset of another (sub)domain or of a constructed domain. A named domain is either a basic domain or a subdomain. *Note* : as we shall see below, a *relation* is a set of similar elements and can therefore be considered as a domain by itself. The following examples define seven representatives subdomains.

```
NAME : char(25)
STREET : NAME
```

```
PERSON, DEPARTMENT, PRODUCT : ENTITIES
WORKMAN, EMPLOYEE : PERSON
```

Constructed domains and domain constructors

A constructed domain is defined by a *domain constructor*, and is a set elements obtained by applying set and/or relational operators on domains, or the elements of which are explicitly listed. There are four domain constructors, namely the *cartesian*, *powerset*, *algebraic*, and *list* constructors.

The ***cartesian constructor*** defines the cartesian product of one or several domains. Each occurrence or partnership of a domain in the product is called an *attribute* of the product. The attributes of the product are given distinct names. When an attribute and its domain are given the same name, the specification of the latter is dropped, leading to a more concise notation. In Examples 2.1, CITY and ADDRESS are subdomains based on cartesian constructed domains. For instance, CITY has two attributes, namely ZIP-CODE, defined on subdomain ZIP-CODE (concise notation) and CITY-NAME, defined on *char* predefined domain.

The ***powerset constructor*** defines the set of subsets of a domain. A constraint is stated about the size of the subsets by declaring their minimum (`min`) and maximum (`max`) size with expression `[min:max]`. Omitting the `min` value means `0`, and ∞ is denoted with `*`. In particular, the notation `[*]` denotes sets of any size. In Examples 2.1, CHR-NAMES, GANG and SET-OF-GANGS are subdomains based on powerset domains. A CHR-NAMES element is a set of 1 to 4 NAME values, while a SET-OF-GANGS element is a (possibly empty) set of GANG elements, which are in turn non empty sets of WORKMAN elements. *Note* : the `[min:max]` notation for powersets will not be confused with the `[attribute-list]` notation for algebraic project constructor.

The ***algebraic constructor*** defines a set of elements as the result of the evaluation of an algebraic expression on domains. The primitive algebraic operators can be used (union, difference, relational product, project, select), together with derived ones (join, intersect, divide, etc). *Note :* the cartesian and powerset constructors are basically algebraic operators; however, due to their importance they are presented separately. CODE, CUSTOMER and OPTIONAL-INTEGER are subdomains based on algebraic constructed domains. In Examples 2.2, MAN and WOMAN subdomains are defined as projections of selected subsets of desc-of-PERSON tuples (i.e. MAN is made up of PERSON elements that appear in desc-of-PERSON tuples where SEX = 'M').

The ***list constructor*** defines a set of elements comprehensively by a list of the denotations of each of them (DAY-OF-WEEK in Examples 2.1). The *null set* comprises one special element called the *null* element, denoted by ∅. It is compatible with (and therefore can be added to) any domain. It should not be confused with the *empty set*, denoted by `{}`.

```
NAME : char(25)
CODE : char(10) ∪ integer
NUMBER, ZIP-CODE : CODE
STREET : NAME
PERSON, DEPARTMENT, PRODUCT : ENTITIES
WORKMAN, EMPLOYEE : PERSON
CITY : (ZIP-CODE,CITY-NAME : char(30))
ADDRESS : (NUMBER,STREET,CITY)
CHR-NAMES : NAME[1:4]
```

```
GANG : WORKMAN[1:*]
SET-OF-GANGS : GANG[*]
CUSTOMER : (PERSON - EMPLOYEE) ∪ DEPARTMENT
DAY-OF-WEEK : {'SUN','MON','TUE','WED','THUY','FRI','SAT'}
OPTIONAL-INTEGER : integer ∪ {∅}
```

Examples 2.1 - *Some subdomain definitions.*

Relation schemata

A relation schema is defined by the cartesian constructor and is given a name. The attributes of the cartesian product are those of the relation schema. Exemples 2.2 proposes some definitions. The *desc-of-PERSON* relation schema associates basic information with each entity PERSON; ACCOUNTS excepted, the attributes and their respective domains are given the same names. MAN and WOMAN are algebraically defined domains on which the MARRIAGE relation schema is defined. The latter is used as a domain in the DIVORCE relation schema.

```
desc-of-PERSON(PERSON,NAME,CHR-NAMES,ADDRESS,SEX,ACCOUNTS: CODE[*])
WOMAN : desc-of-PERSON(SEX = 'F')[PERSON]
MAN : desc-of-PERSON(SEX = 'M')[PERSON]
MARRIAGE(wife: WOMAN, husband: MAN, DATE)
SALES(salesman: EMPLOYEE, buyer: CUSTOMER, PRODUCT, SALES-DATE: date)
DIVORCE(MARRIAGE,DATE)
LINE(PRODUCT,QTY: integer)
desc-of-ORDER(ORDER,ORDNUM,DATE,CUSTOMER,ADDRESS,LINES: LINE[1:20])
```

Examples 2.2 - *Some relation schema definitions.*

It is interesting to note that the notion of subdomain (and specially *entity* subdomain) makes unary relation schemata practically useless.
One can observe that the notion of relation schema (declared with expression *`rel-name(attribute-def-list)`*) is redundant with that of cartesian subdomain (declared with expression *`subdomain-name : (attribute-def- list)`*). However, both concepts (and notations) are kept due to the relational origin of the current concepts. Moreover, let's remember that our purpose is not to define a new relational model but to lay down a formal framework to describe, compare and cross-translate E-R models.

Constraints

We shall mention three classes of constraints that will be of particular importance in the following, namely the *key* constraints, the *dependency* constraints and the *inclusion* constraints.

The ***key*** **constraint** is defined on a relation schema and is to be evaluated on any of its instances. It expresses the uniqueness of t-uples according to the values of a subset of attributes or of components of attributes. It should be reminded that each value of a powerset attribute that participates in a key is a value set. If elements of these value sets are meant, the [*] notation will be used, as in the ACCOUNTS[*] key in Examples 2.3, stating that no two *desc-of-PERSON* can share the same CODE value in their ACCOUNTS value sets. The relation schema can be an explicitly declared, or it can be a derived relation schema defined by an algebraic constructor. When no ambiguity can arise, the underlined notation will be used. In the sample schemata, we shall follow the following convention : a key will be specified only if it fully determines all other attributes and if it does not comprise all the attributes. In Examples 2.3, a *desc-of-PERSON* t-uple is identified **either** by its PERSON element **or** by its NAME value, its set of CHRISTIAN-

NAME values and its ADDRESS value, **or** by any element of its ACCOUNTS set (not only by the whole set, as will be stated without the [*] specification). *DIVORCE* t-uples have distinct MARIAGE t-uples. In any instance of the LINE[1:20] set appearing as LINES in a *desc-of-ORDER* tuple, all t-uples have distinct PRODUCT elements. The last example illustrates the underlined notation of the first two keys of *desc-of-PERSON*.

```
key(desc-of-PERSON) : PERSON
key(desc-of-PERSON) : NAME,CHR-NAMES,ADDRESS
key(desc-of-PERSON) : ACCOUNTS[*]
key(DIVORCE) : MARRIAGE
key(desc-of-ORDER[LINES]) : PRODUCT
desc-of-PERSON(PERSON,NAME,CHR-NAMES,ADDRESS,SEX,...)
```

Examples 2.3 - *Key constraint definitions*

The ***dependency*** **constraint** is defined on a relation schema and is to be evaluated on any of its instances. It states that a functional, multivalued or more generally join dependency always holds in these instances. The relation schema can be an explicitly declared or a derived relation schema defined by an algebraic constructor. In the sample schemata, we shall follow the following convention : a functional dependency will be specified only if it is not implied by the keys. Examples 2.4 gives some illustrations of this concept. The first two definitions (1) apply on schemata *SALES* and *desc-of-ORDER* from Examples 2.2. The last example (3) is a revised illustration of the well-known example of the lost dependency problem in some BCNF decompositions [DATE,86].

```
SALES : buyer,SALES-DATE ⟶ salesman                (1)
desc-of-ORDER : CUSTOMER ⟶ ADDRESS
PROSPECT(SALESMAN,REGION,PRODUCT)                  (2)
PROSPECT : REGION ↠ PRODUCT
TJ(TEACHER,SUBJECT)                                (3)
ST(STUDENT,TEACHER)
ST*TJ : STUDENT,SUBJECT ⟶ TEACHER
```

Examples 2.4 - *Functional and multivalued dependency constraint definitions.*

The ***inclusion*** **constraint** is an inter-domain constraint stating that a (generally constructed) domain is a subset of (**"**), or equals (=) to, another one. Some definitions can be found in Examples 2.5. The first example complements the *TJ-ST* schema in Example 2.4 by stating that a STUDENT may only be related to TEACHERS who teach a SUBJECT. The second one precises that any known PERSON must be described (i.e. any PERSON element must participate in a *desc-of-PERSON* t-uple).

```
ST[TEACHER] ⊆ TJ[TEACHER]
desc-of-PERS[PERSON] = PERSON
SUPPLY(CUSTOMER,PRODUCT,STORE,DATE)                (1)
AVAILABILITY(PRODUCT,STORE,DATE)
SUPPLY[PRODUCT,STORE,DATE] ⊆ AVAILABILITY
```

Examples 2.5 - *Inclusion constraint definitions.*

The last example (1) states that one may supply a PRODUCT from a STORE on a given DATE only if that PRODUCT is available from that STORE on that DATE.

Referential constraint is another popular version of inclusion constraint; however, the concept of entity subdomain makes it practically useless in most cases.

NOTE. A complete definition of the model should state the dynamic rules that are to be applied in order to preserve integrity constraints in case of update. However, since static aspects only are concerned, any updating strategy that preserves integrity is acceptable.

3. THE BASIC E-R MODELS AND THEIR RELATIONAL EXPRESSION

Distinguishing basic models from extended models is rather arbitrary. A feature can be called *basic* when it is included in a significant number of published and/or used (in practical methodologies and CASE tools) E-R models. This section recalls the main concepts that can be found in basic E-R models. Each of them will be given an extended relational expression that formalizes it while conveying the same semantics.

3.1 Entity type

An *entity type* stands for a class of autonomous, remarkable and worth being distinguished, abstract or concrete objects in the context of the UoD. As already quoted in [HALL,76], that notion still resists any attempt of axiomatisation. An *entity type* E will be represented by an entity domain E (Schema 3.1). Each entity domain element is a *surrogate* [HALL,76] for a UoD *entity*.

```
CUSTOMER : ENTITIES
ORDER : ENTITIES
```

Schema 3.1- *Relational expression of entity types CUSTOMER and ORDER*

3.2 Attribute of an entity type

An *attribute* A_i of entity type E describes a simple property (i.e. not worth being described as an entity type) that characterizes each E entity. A *descriptive relation schema* named, for instance, desc-of-*E* is defined on the entity domain E - through an *entity attribute* with name E - and on relational attributes corresponding to attributes A_i of E. Entity attribute E is, by definition, a relational key of descriptive schema desc-of-*E* (Schema 3.2).

```
desc-of-CUSTOMER(  CUSTOMER,
            CUS-NUM : num(6),
            CUS-NAME : char(25),
            CUS-ADDRESS : char(40),
            CUS-ACCOUNT : num(10) )
   desc-of-ORDER(  ORDER,
            ORDER-NUM : integer,
            DATE : date)
```

Schema 3.2- *Relational expression of the attributes of entity types CUSTOMER and ORDER.*

```
NAME : char(10),
STD-ADDRESS : ( NUMBER: integer,STREET: NAME,CITY: NAME),
desc-of-PERSON( PERSON,
            PNAME : char(30),
            CHR-NAME : NAME[1:4],
            ADDRESS : STD-ADRESSE,
            SPOUSE-NAME : char(30) ∪ {∅} )
```

Schema 3.3- *Relational expression of complex entity type attributes*

E-R *multivalued* attributes are represented by relational attributes based on a powerset domain (CHR-NAMES values in Schema 3.3); representation of E-R *compound* attributes are based on cartesian domains (ADDRESS in Schema 3.3). E-R optional attribute expressions will be defined on domains which include the *null element* (SPOUSE-NAME in Schema 3.3), or that can be defined as multivalued attributes with cardinality [0:1] (*optional* attributes can be got rid of by decomposing the descriptive relation schema).

NOTE. Decoupling the relational specification of an entity type from that of its attributes allows the definition of entity types without attributes.

3.3 Relationship type

A *relationship* (*type*) is defined by a collection of 2 or more entities (entity types), each of them playing a specific *role* in this collection. A relationship type R, with roles ri played by entity types Ei is represented by an *associative relation schema* R with attributes ri defined on entity domains Ei. For each attribute Aj of relationship type R, a relational attribute Aj is defined in relation schema R.

```
SENDS ( SENDER: CUSTOMER, ORDER)
SUPPLY( SUPPLIED-ORDER: ORDER,
      SUPPLIED-PRODUCT: PRODUCT,
      SUPPLIER,
      SUPPLIED-QUANTITY: num(5))
```

Schema 3.4 - *Relational expression of relationship types*

A more concise representation can be proposed, particularly when R is functional (*N-1*) and has no attributes, by *joining* its associative relation schema with the descriptive relation schema of the *image* entity type. In Schema 3.5, *desc-of-ORDER'* is obtained by joining *SENDS* (from Schema 3.4) with *desc-of-ORDER* (from Schema 3.2). The join is legal if `SENDS[ORDER] = desc-of-ORDER[ORDER]` (the relational schema, {`D(E1,A1,..,An); R(E1,E2); R[E1]=S[E1]`} can always be replaced by schema {`D'(E1,A1,..,An,E2) = D*S`}, which is lossless and preserves dependencies [HAINAUT,88]).

```
desc-of-ORDER'( ORDER,
      ORDER-NUM: ..,
      DATE: ..,
      SENDER:  CUSTOMER)
```

Schema 3.5 - *Concise relational expression of a functional relationship type*

3.4 Identifier of an entity type

A set {Ai,..,Aj} of one or several attributes of entity type E is an identifier for E if no two E entities can have the same value(s) for these attributes. That identifier is described by a candidate key of the descriptive relation schema that is made up of attributes Ai,..,Aj. *Hybrid* or *local* identifiers, using an *identifying relationship type* [CHEN,76], can be described easily in the concise expression of functional relationship types as can be seen in Examples 3.6. Let's consider statistics on *monthly sales* of each product. A MONTHLY-SALES entity is identified by both the PRODUCT entity with which it is associated and the DATE on which the sales of that product are evaluated. Hence the descriptive relation schema *desc-of-MONTHLY-SALES*, with identifier {PRODUCT, DATE}. *NOTE*. An entity type with a hybrid identifier is sometimes called a *weak entity type*.

```
desc-of-CUSTOMER(CUSTOMER, CUST-NUM, CUST-NAME, etc)
desc-of-MONTHLY-SALES(MONTHLY-SALES, PRODUCT,DATE, QTY)
```

Schema 3.6 - *Simple and hybrid entity type identifiers*

3.5 Cardinality constraint of a role

The cardinality constraint of role rk played by entity type Ek in relationship type R is defined by a couple i-j of integers stating that any Ek entity must play role rk in m relationships R, such that i≤m≤j. By giving the entity attribute the name of the role rk it represents, that constraint translates as follows in the relational representation,

$$\forall e \in E_k,\ i \le |R(r_k = e)| \le j$$

The most common values of i-j are 0-1, 1-1, 0-∞, 1-∞. In these cases, simpler expressions can be proposed as follows.

- If i = 1, any element of entity domain Ek must appear at least once as attribute rk in any relation R, so that :
  $$E = R[r_k]$$
- If i = 0, that constraint does not stand.

- If j = 1, any element of entity domain Ek can appear no more than once as attribute rk in any relation R, so that attribute rk is a candidate key of relation schema R :
  $$R(r_1, .., \underline{r_k}, .., r_n)$$
- If j = ∞, the latter constraint does not stand.

An E-R schema SUPPLY in which

ORDER plays the role supplied-order with cardinality 1-1,
PRODUCT plays the role supplied-product with cardinality 0-∞,
SUPPLIER plays the role supplier with cardinality 0-∞,

is losslessly translated into schema 3.7.

```
SUPPLY( SUPPLIED-ORDER: ORDER,
    SUPPLIED-PRODUCT: PRODUCT,
    SUPPLIER: SUPPLIER,
    SUPPLIED-QTY: num(5))
ORDER = SUPPLY[SUPPLIED-ORDER]
```

Schema 3.7 - *Relational expression of cardinality constraints*

It is worth noticing an important constraint on the properties of a relationship type R intervening in a hybrid identifier of entity type E1 (see section 3.4). Roughly speaking, R must be functional when E1 is considered as the source domain. More precisely, let R be binary and have roles r1 played by E_1 and role r2 played by E2, not necessarily distinct from E1 (i.e. R(r1:E1,r2:E2)). R can be a component of the hybrid identifier of E1 if cardinality constraints of r1 are 1-1. It is now easy to understand the very reason of these constraints. They are the necessary conditions that allow the translation of the hybrid identifier into a relational key. If r1 is i-∞, R cannot be translated into a single-value entity attribute of relation schema desc-of-E1. If r1 is 0-j, entity attribute r2 is optional (i.e. *null* allowed). In that case, the *entity integrity* principle is violated from the strict relational viewpoint [DATE,86b].

4. RELATIONAL EXPRESSION OF EXTENDED E-R MODELS

The basic concepts of section 3 have long been considered as too poor a model to satisfy the increasing complexity of the target UoD's. That observation developed, among others, from two sources, namely the relational theory itself (including recent developments), and other domains devoted to semantics representation, particularly in AI [WOODS,75]. We shall analyse the most prominent enrichments that are available in currently used extended E-R models, and we shall propose a relational expression that captures their whole semantics. However, that analysis does not attempt to build a consistent and non redundant set of concepts. The most comprehensive and in-depth analysis of the semantic extensions of the E-R models can be found in [COLLART,88].

4.1 Dependencies

Some E-R models provide ways to describe dependencies other than those inferred from the entity type identifier(s) [BODART,88]. The main extensions concern dependencies between attributes of an entity type, between attributes of a relationship type and between roles of a relationship type. Their relational expression is immediate (Schema 4.1).

```
desc-of-EMPLOYEE(EMPLOYEE,EMP-NUM,EMP-NUM,DEPART-NAME,ADDRESS)
SUPPLY(ORDER,PRODUCT,SUPPLIER,QTY)
SALES(SALESMAN,REGION,PRODUCT)
desc-of-EMPLOYEE: DEPART-NAME ⟶ ADDRESS
SUPPLY: ORDER,PRODUCT ⟶ SUPPLIER
SALES: REGION ↠ PRODUCT | SALESMAN
```

Schema 4.1 - *Representation of dependencies in relationship types*

4.2 Inclusion and exclusion constraints

These constraints are associated with roles of relationship types. Let R1, R2 be relationship types such that ,

- R1 has roles r11 played by E1, .., r1i played by Ei, .., r1n played by En;
- R2 has roles r21 played by E1, .., r2i played by Ei, .., r2m played by Em.

```
SUPPLY(SUPPLIER,PRODUCT,PRICE)
ASSIGN(ORDER,PRODUCT,SUPPLIER,ASS-QTY)
desc-of-INVOICE(INVOICE,INV-NUM,DATE,ORDER,..)
ASSIGN[PRODUCT,SUPPLIER] ⊆ SUPPLY[PRODUCT,SUPPLIER]   (a)
descr-of-INVOICE[ORDER] = ASSIGN[ORDER]               (b)
```

Schema 4.2 - *Relational expression of inclusion constraints*

An *inclusion constraint* states than for any R1 relationship r, the E1,..,Ei entities playing roles r11,..,r1i in r must also play roles r21,..,r2i in at least one R2 relationship. That constraint is translated into an inclusion constraint between projections of the corresponding associative relation schemata. According to statement (a) in Schema 4.2 assignment of a PRODUCT to a SUPPLIER is allowed only if that SUPPLIER can supply that PRODUCT. The *equality constraint* is a particular case in which the converse constraint also holds. Statement (b) of Schema 4.2 expresses that an INVOICE is issued once there is at least one assignment for the ORDER, and conversely. When i is 1, the constraint is generally perceived as associated with entity type E_1, while when i is greater than 1, it is perceived as a constraint on relationship type R1.

An *exclusion constraint* between entity-compatible role r11 of relationship type R1 and role r21 of relationship type R2 states that if an entity plays at least once role r11, then it cannot play role r21 in any R2 relationship. Some other constraints related to authorized combinations of roles and/or attributes are to be found in some extended models. Schema 4.3 proposes an expression for some of them. According to (a), each RECORD is associated with an EMPLOYEE or with a WORKER (and possibly both); according to (b), a RECORD cannot be associated with both a WORKER and an EMPLOYEE.

```
RECORD-of-EMPLOYEE(RECORD,EMPLOYEE)
RECORD-of-WORKER(RECORD,WORKER)
RECORD = RECORD-of-EMPLOYEE[RECORD] ∪ RECORD-of-WORKER[RECORD]   (a)
RECORD-of-EMPLOYEE[RECORD] ∩ RECORD-of-WORKER[RECORD] = {}       (b)
```

Schema 4.3 - *Relational expression of some other role constraints.*

4.3 Extended entity type identifier

Three extensions are proposed, namely more than one identifier, no identifier at all and generalized hybrid identifier. Representing entity types without identifier raises no problem since any descriptive relation schema has at least one key, made up of the described entity attribute. The notion of hybrid identifier can be generalized by allowing such a structure to accomodate more than one relationship type component, and no attribute component. Schema 4.4 gives the relational expression of some extensions.

```
desc-of-LINE-OF-ORDER(LINE-OF-ORDER, ORDER,PRODUCT,ORD-QTY)
desc-of-EMPLOYEE(EMPLOYEE, corp-EMP-ID, local-EMP-ID,DEPARTMENT,etc)
desc-of-LINE-OF-INVOICE(LINE-OF-INVOICE, INVOICE,LINE-OF-ORDER, QTY)
```

Schema 4.4 - *Relational expression of some extensions of entity type identifiers.*

It is interesting to observe that the described entity attribute of a descriptive relation schema, being a candidate key (in fact it can be considered the primary key), a transitive dependency may hold in a descriptive relation schema that has no other key. Therefore the notion of *normal form* of an entity type must be generalized accordingly.

4.4 Relationship type identifier

Relationship type identifier is a less frequent concept. Except in the case of roles with cardinality constraint `i-1`, where any entity can play that role in at most one relationship, the explicit or implicit hypothesis is that no two relationships of a given type can be defined on the sames entities, playing the same roles. These concepts are thoroughly translated in a relational key made up of the entity attributes(s) representing the concerned role(s). Generalization of that notion is sometimes proposed, in which the identifier is made up of a subset of the participating entity types only, or in which an identifier can comprise attributes. Schema 4.5 proposes a translation for some extended identifiers. In these examples, PLANT, PRODUCT, ORDER and SUPPLIER are entity attributes.

```
PRODUCTION(PLANT, PRODUCT,QTY)
TOTAL-SUPPLY(ORDER,PRODUCT,SUPPLIER,SUPPLIED-QTY)
DAILY-SUPPLY(ORDER,PRODUCT,SUPPLIER,DATE,SUPPLIED-QTY)
ASSIGN(ORDER,PRODUCT,SUPPLIER,ASS-QTY)
LINE-of-ORDER(ORDER,LINE-NUMBER,PRODUCT,ORD-QTY)
```

Schema 4.5 - *Relational expression of some relationship type identifiers*

4.5 Generalisation/specialisation of entity types

The concepts of generalization and of specialisation, leading to *supertype/ subtype hierarchies*, are of particular importance as far as conceptual modeling is concerned. Developed mainly in semantics representation techniques [WOODS,75], they have been adopted in the framework of the relational theory [SMITH,77] [CODD,79], then in the E-R model by numerous authors and practitioners (it was suggested, but not developed in [CHEN,76]). The main interest of these concepts is the inference rule based on the inheritance mechanism that states that, in a supertype/subtype hierarchy, any entity of the subtype inherits all the properties that are associated with each of its supertypes. The most straightforward relational expression of a supertype/subtype hierarchy is the entity subdomain, as illustrated in Schema 4.6, in which MAN and WOMAN can be interpreted as specializations of PERSON.

```
PERSON : ENTITIES
MAN : PERSON
WOMAN : PERSON
desc-of-PERSON(PERSON,PERS-NUM,PERS-NAME,PERS-ADDRESS,SEX)
desc-of-MAN(MAN,MILITARY-STATUS)
MARRIAGE(MAN,WOMAN,DATE)
RECORD(PERSON,SKILL)
```

Schema 4.6 - *Relational expression of a specialization structure*

In some cases, a specialization condition can be stated to precise the common specific properties of the members of a subtype. The definitions of MAN and WOMAN in Examples 2.2 are examples of such conditions. Specializations `{MAN,WOMAN}` forms a *covering* for `PERSON` if `PERSON` = `MAN` $\cup$ `WOMAN`. If `MAN` $\cap$ `WOMAN` = `{}`, specializations `{MAN,WOMAN}` forms a *disjunction*. A set of specializations that is both a covering and a disjunction is a *partition* of the supertype. The supertype of a partition is often called its *generalization*. An entity type `E` can be a specialization of more than one supertype `Ei`. This can be expressed by defining `E` as a subset of each `Ei`.

The inheritance mechanisms will be given a relational interpretation as follows. Top-down inheritance can be materialized with a *natural join* of the descriptive relation schemata. The *complete* description of MAN is obtained as follows :

$$\text{desc-of-MAN} * \text{desc-of-PERSON}$$

Bottom-up inheritance can be dealt with by a *right (or symmetric) outer natural join* (according to [DATE,86b] notation). A *complete* description of PERSON is obtained as follows :

$$\text{desc-of-MAN} \Lambda * \text{desc-of-PERSON}$$

Let's observe, however, that the join and outer join operators define inheritance of attributes and of functional relationship type partnership. Propagating the partnership in other kinds of relationship types must be expressed for each of them as exemplified in Schema 4.7. By analogy with some hybrid knowledge representation techniques, partial, default and redefined inheritance can be defined [COLLART,88]. The latter reference proposes the concept of *lateral inheritance* between overlapping subtypes.

```
PERSON : ENTITIES
MAN : PERSON
WOMAN : PERSON
desc-of-MAN'(PERSON,PERS-NUM,
        PERS-NAME,PERS-ADDRESS,SEX,MILITARY-STATUS)
MARRIAGE(MAN,WOMAN,DATE)
MAN-RECORD(MAN,SKILL)
```

Mapping : `desc-of-MAN' = desc-of-MAN * desc-of-PERSON`

Mapping : `MAN-RECORD = RECORD[MAN,SKILL]`

Schema 4.7 - *Materialization of top-down inheritance mechanisms applied to entity type MAN as described in Schema 4.6.*

4.6 Generalisation/specialisation of relationship types

The specialization/generalization structure is sometimes applied to relationship types [JUNET,87]. This is quite valid since such structures are defined between sets. The relational expression is by subsetting constraints between associative relations.

4.7 Category

The category concept, as defined by Elmasri, Weeldreyer and Hevner [ELMASRI,85] (also suggested in [CODD,79]) is an extension of the concept of generalization. In relational terms, a category is simply described by a new entity subdomain defined as a subset of the union of (one or) several entity domains. Schema 4.8 states that a member of the category CUSTOMER is a PERSON or a COMPANY.

$$\text{CUSTOMER} : \text{PERSON} \cup \text{COMPANY}$$

Schema 4.8 - *Relational expression of the category CUSTOMER*

4.7 Grouping aggregation

This complex notion was proposed in [HAMMER,81], [CODD,79], [MEES,87] among others. It consists in considering a subset of entities as an entity of its own. Schema 4.9 is the relational expression of a situation derived from that proposed in [CODD,79]. CONVOY represents a homogeneous aggregation and CONVOY' a heterogeneous one.

```
SHIP : ENTITIES
CONVOY : SHIP[*]
CONVOY' : (SHIP ∪ PLANE ∪ OFFICER)[*]
```

Schema 4.9 - Relational expression of *grouping aggregation*

4.9 Multidomain role

This notion is rarely mentioned [HAINAUT,81], in spite of its considerable usefulness. Moreover, it is available in some popular DBMS (as *multi-member* and *multi-owner* set types in CODASYL-like DBMS or in MDBS-3). A multidomain role is defined on the union of entity domains instead of on one domain. In the example of Schema 4.9, role OWNER can be played by an EMPLOYEE or a SERVICE. By comparing schema 4.10 and the category expression, we can observe the closeness of both concepts.

```
PROPERTY(VEHICLE: CAR, OWNER: EMPLOYEE ∪ SERVICE)
```

Schema 4.10 - *Relational expression of a multidomain role*

4.10 Multivalued and optional role

It is a fairly rare proposal [JUNET,87], probably inspired by the relational similarities between the concepts of role and attribute. A role `ri` of a relationship type R can be played by a set of entities instead of by one entity. Let's suppose a UoD where gangs of workmen are in charge of repairing failures in various types of equipment. A natural formalization could be by a relationship type REPAIR between FAILURE entities and sets of WORKMEN. The relational interpretation is given in schema 4.11.

```
REPAIR(FAILURE, GANG: WORKMAN[1:*])
```

Schema 4.11 - *Relational expression of a multivalued role*

The similarity with the *grouping aggregation* concept is obvious. A natural extension is the *optional* role, which can be void in some relationships.

4.11 Role defined on a relationship type domain

That concept, sometimes called *relationships between relationships*, allows a relationship itself to be associated with foreign entities and other relationships [SCHEUERMANN,80], [MEES,86], [JUNET,87]. Such a construct can induce a great economy of specification (by avoiding artificial entity types) and is a way to solve the problem of incomplete information (optional role is another way). Schema 4.12 gives the relational expression of the `LINE-of-INVOICE` concept, which is declared as a relationship type between entity type `INVOICE` and relationship type `LINE-of-ORDER`.

```
ORDER,PRODUCT,INVOICE : ENTITIES
LINE-of-ORDER(ORDER,PRODUCT,ORD-QTY)
LINE-of-INVOICE(INVOICE,LINE-of-ORDER,INV-QTY)
```

Schema 4.12 - *Relational expression of a relationship type role*

4.12 Complex objects

A complex, or structured, object [ABITEBOUL,87] [HULL,87] is made up of values and/or other objects. It can be seen as an entity, some attribute values of which are entities. In current models, that concept is formalized by N1NF or Nested relations, or by special relationships such as *part-of* between objects (e.g. in Knowledge representation techniques). A traditional example describes *parts* that can be made up of other *parts*. Schema 4.13 suggests two relational expressions of that problem. In the first expression (a), emphasis is put on the relationships betwen parts, while expression (b) recursively defines a part as a collection of other parts.

```
PART : ENTITIES
PART-SUBPART(PART,SUBPART: PART[0:*])              (a)
def-of-PART(PART,COMPOSITION: def-of-PART[0:*])    (b)
```

Schema 4.13 - *Relational expression of a complex object*

4.13 Binary models

Binary models (and more generally irreducible models) can be compared with E-R models as far as semantic description of the UoD is concerned. One of the most popular binary models for conceptual design is that of NIAM [VERHEIJEN,82]. Its relational expression raises no particular problem. The concepts can be translated as follows : a NOLOT is represented by an entity subdomain, a LOT by a simple value domain, naming bridges and ideas by binary relationship types, roles by attributes. Concepts such as subtype, uniqueness, totality, exclusion, subsetting, etc, are thoroughly represented in the extended relational model. To stick more closely to the NIAM terminology and structures, some specific conventions can be adopted as to the building of descriptive and associative relation schemata. For instance, a NIAM relationship has no name, but its two role names have a particular importance. Therefore a rule for systematically naming relation schemata has to be adopted.

4.14 Exceptions : multisets and instance/class duality

Among the most important E-R extensions, we have to mention the notion of ***multiset*** that allows the definition and manipulation of collections in which an element can appear more than once. Using *multisets* is proposed by Parent and Spaccapietra [PARENT,86]. Despite its theoretical and practical interest, this extension has not been kept here nor expressed in the relational formalism adopted, due to the set hypothesis of the latter.

According to the *classification* mechanism which is at the very basis of data base models, an *object type* (or *class*) is an autonomous concept that is meant for abstracting any set of similar objects, called its *instances*. Classes and instances lives in distincts abstraction layers between which one relationship only is allowed (**instance-of**). In some application domains, more flexible modeling structures are needed. The following example will illustrate that point. Let's consider an application data base about customers,

with an associated data dictionary. According to that structure, we can describe a customer called 'Smith'. So, 'Smith' **is** (the name of) **a** CUSTOMER (described in the data base), and CUSTOMER **is an** ENTITY-TYPE (described in the data dictionary). However, 'Smith' **is not an** ENTITY-TYPE, showing that the **is a** relationships used here cannot be composed. A more precise analysis leads to the following formalization : 'Smith' **is-an-instance-of** CUSTOMER, and CUSTOMER **is-an-instance-of** ENTITY-TYPE. Therefore, CUSTOMER is both an instance and a class depending on the context in which it is considered. *Instance* and *class* are not intrinsic properties of an object, but roles it can play in various contexts. Very few models allow for this duality of roles. KEE's model is one of them [FIKES,85]. The model proposed in this paper is basically a data base model, and therefore has not been extended to accomodate that feature.

5. GER : A GENERIC ENTITY-RELATIONSHIP MODEL

As already mentioned, sections 3 and 4 were not intended to propose a comprehensive and consistent set of E-R extensions. Indeed, seen as a whole, these extensions quickly appear to be complex, not consistent, redundant, irregular and incomplete. On the other hand, many relational expressions have no direct E-R counterpart (see the DIVORCE relation in Examples 2.2). From that analysis and translation exercise, we can now propose a redefinition of an E-R model the constructs of which encompass the extensions described above while remaining both simple and non redundant. These definitions were developed by subsetting the extended relational concepts according to the semantic variety of the E-R approach. One of the objectives was that each E-R construct would have one and only one relational expression, and conversely. That one-to-one mapping between both expressions makes them equivalent. It is worth noticing that such a mapping is not identical to the various *E-R-to-relational* transformations that are proposed in DB design methodologies [TEOREY,86], and in which semantics degradation is accepted. The resulting model is called the *Generic Entity-Relationship* (GER) model. Quite naturally, the GER model is both less powerful and a bit more complex than its relational origin. The syntax of a definition language for GER schema specification is suggested. We shall propose no example of GER schemata in this section since the examples of sections 3 and 4 totally comply with the syntax proposed.

The following definitions are intended to give a fairly precise but still intuitive perception of the GER model. They cannot be considered fully formalized.

5.1 Entity domains

- An **entity** is an element of the *universal* entity domain called ENTITIES. Entities are created and can be deleted. At any given time, all entities are distinct.
- A new **entity domain** E2 can be defined so that each of its elements, at any time, belongs to entity domain E1. E2 is called a *subdomain* of E1 and E1 a *superdomain* of E2. If E2 is explicitly declared as a *subdomain* of E1, it is a *direct subdomain* of E1, and the latter is also a *direct superdomain* of E2. Inference rule : if E3 is a subdomain of E2 and E2 is a direct subdomain of E1, then E3 is a subdomain of E1. An entity domain can be a subdomain of more than one superdomain. A direct subdomain of the universal entity domain is called a *basic entity domain*. An entity domain can be declared a subset of a *constructed domain* obtained by applying powerset, union, intersection, difference and projection operators. Entity domains are given distinct names.
- At any time, an entity can enter or leave a subdomain of its basic entity domain. The latter, however, is unique and fixed during the life of the entity.

- All (past, present and future) entities that are in an entity domain are said to be of a given *entity type*. That type is defined by all the structural and behavioural properties that are common to its entities.

Suggested syntax

ET-decl ::= ET-name **:** ET-domain-designation
ET-domain-designation ::= **ENTITIES** | ET-name | ET-domain-constructor | **{}**
ET-domain-constructor ::= ET-powerset-constr | ET-algebraic-constr | {∅}
ET-powerset-constr ::= ET-name **[** range **]**
ET-algebraic-constr ::= union | intersection | difference | project *resulting in Entity sets*

5.2 Value domains

- A **value** is any permanent symbol that can be stored, transmitted and processed in a manual or automated information processing system. A value has a type which defines the set of possible values, the properties and the processing rules of the values.
- A **value domain** is a named set of values of a given type.
- There exist predefined *basic value types*, namely integer, real, character string, date, etc. The *basic value domain* of a basic type is the set of all the possible values of that type.
- A new value domain can be defined as a subset of an existing one or as a subset of a constructed value domain. The legal constructors are the cartesian, powerset and list constructors.
- A *simple value domain* is either a basic value domain or a subset of a simple value domain. Constructed value domains are called *complex value domains*.

Suggested syntax

V-subdomain-decl ::= V-subdomain-name **:** V-domain-designation
V-domain-designation ::= V-domain-name | V-domain-constr
V-domain-name ::= V-subdomain-name | V-basic-domain-name | V-relation-name
V-basic-domain-name ::= **integer** | **real** | **char(**integer**)** | **date** | *etc*
V-domain-constr ::= V-cartesian-constr | V-powerset-constr | V-list-constr
V-cartesian-constr ::= **(** V-attribute-def-list **)**
V-attribute-def-list ::= V-attribute-def | V-attribute-def **,** V-attribute-def-list
V-attribute-def ::= attribute-name **:** V-domain-designation | V-subdomain-name
V-powerset-constr ::= V-subdomain-name **[** range **]**
V-list-constr ::= **{**value-list**}**
value-list ::= value | value **,** value-list

5.3 Entity relation schemata

An **entity relation schema** is the descriptive relation schema that defines the E-R attributes of the entity type and its partners (i.e. entity types playing a role) in some functional - or *many to one* - relationship types. The described entity type is a GER attribute of the entity relation schema. That attribute is called the *described entity attribute* and is based on the *described entity domain*. It is the primary key of the relation schema. One suggests the specific syntax "`desc-of-`*`ET-name`*" for descriptive entity relation schemata. However, more than one descriptive relation schema can be defined for each entity type (with ad-hoc naming conventions).

Suggested syntax

ET-rel-schema ::= ET-rel-decl ET-key-list
ET-rel-decl ::= ET-rel-schema-name **(** ET-name**,** role-attribute-def-list **)**
ET-rel-schema-name ::= **desc-of**-ET-name | *any name*
role-attribute-def-list ::= role-attribute-def | role-attribute-def **,** role-attribute-def-list
role-attribute-def ::= attribute-def | role-def
attribute-def ::= attribute-name **:** V-domain-designation
role-def ::= role-name **:** E-set-designation | ET-name
E-set-designation ::= ET-name | ET-name ∪ {∅}

5.4 Relationship relation schemata

A relationship is an aggregate of at least two entities and/or relationships, together with any number of values. Any relationship belongs to a relationship type that is defined by all the structural and behavioural properties that are common to its relationships. A *relationship relation schema* describes the roles and the E-R attributes of a relationship type the description of which has not been included in an entity relation schema. At least two GER attributes must be defined on entity domains and/or on other relationship relation schemata. Its name is that of the relationship type described. Note that cardinality constraints have been limited to the most useful values as proposed in 3.5. These values can be losslessly translated into key and identity constraints.

Suggested syntax

RT-rel-schema ::= RT-rel-decl RT-key-list
RT-rel-decl ::= RT-name **(** role-def-list attribute-def-list **)**
role-def-list ::= RT-role-def RT-role-def *possibly empty list of* RT-role-def
RT-role-def ::= role-name **:** ER-domain-designation | ET-name
ER-domain-designation ::= ER-set-designation | ER-set-designation **[**range**]**
ER-set-designation ::= ER-designation | ER-designation ∪ {∅}
ER-designation ::= ET-name | RT-name
attribute-def-list ::= *possibly empty list of* attribute-def

5.5 Constraints

Keys. Any entity and relationship relation schema has at least one key. A component of a key is an attribute, or an attribute of a cartesian attribute, or an element of a powerset attribute. When possible, a key can be specified by the underlined notation.

Dependency. Between any two subsets of attributes, attributes of a cartesian attribute, or elements of a powerset attribute.

Inclusion. Inclusion and identity between any two entity sets, value sets or relations. Entity sets are entity domains or projections of entity- or relationship relation schemata (or of algebraically constructed relation schemata). Value sets are defined by projection of entity or relationship relation schemata (or of algebraically constructed relation schemata). Relations are entity- or relationship relation schemata (or algebraically constructed relation schemata).

Suggested syntax

constraint-decl ::= key-decl | dependency-decl | inclusion-decl | *etc*

ET-key-list ::= **key(**ET-rel-schema-name**) :** ET-name *possibly empty list of* ET-role-attr-key-def
ET-role-attr-key-def ::= **key(**ET-rel-schema-name**) :** ET-role-attribute-list

ET-role-attr-list ::= ET-role-attr-designation | ET-role-attr-designation **,** ET-role-attr-list
ET-role-attr-designation ::= role-name | attr-designation
attr-designation ::= attr-id | attr-id**.**attr-designation
attr-id ::= attribute-name | attribute-name**[*]**

RT-key-list ::= *possibly empty list of* RT-role-attr-key-def
RT-role-attr-key-def ::= **key(**RT-name**) :** RT-role-attr-list
RT-role-attr-list ::= RT-role-attr-designation | RT-role-attr-designation **,** RT-role-attr-list
RT-role-attr-designation ::= RT-role-attr-id | RT-role-attr-id**.**RT-role-attr-designation
RT-role-attr-id ::= RT-role-attr-name | RT-role-attr-name**[*]**
RT-role-attr-name ::= role-name | attribute-name

dependency-decl ::= rel-designation **:** role-attr-list dependency-op role-attr-list
rel-designation ::= ET-rel-schema-name | RT-name | *project and join based algebraic constructor*
dependency-op ::= $\longrightarrow$ | $\rightarrow\!\!\rightarrow$

inclusion-decl ::= domain-designation inclusion-op domain-designation
domain-designation ::= ET-domain-designation | V-domain-designation
inclusion-op ::= $\subseteq$ | **=**

6. EXTENSION TO THE GER MODEL FOR IMPLEMENTATION DESIGN

During the seventies, the frontier between conceptual and implementation (i.e. DBMS-dependent) specifications was not always clear. One of the merits of the E-R approach (and of some other, such as functional and binary approaches) was to clarify the roles of these specification levels. The design activities at the implementation level need a descriptive formalism that allows the expression of conceptual structures **and** the specification of access structures satisfying the performance requirements of applications. DBMS models are sometimes used, but a more general DBMS-independent implementation model is often preferred, particularly in DB-oriented CASE (Data Structure Diagrams is still one of the most popular data models). The theoretical and practical benefits of such a unique model are obvious when considering the complexity of designing DB structures that are both correct and efficient for several target DBMS. However, current practices are not satisfying due to the poverty of the implementation data model used. On the one hand, some existing DBMS models are more powerful than the implementation model used; let's only mention N-N, multi-owner and multi-member set types of MDBS-3 (MDBS Inc.). On the other hand the emergence of E-R and structured-object DBMS, while reducing the gap between the conceptual and DBMS models, leaves the implementation design problem unsolved. The need for a general model for specification of DBMS data structures is therefore clear.

Each DBMS model can be layered into three subsets of concepts, namely the (generally limited) *conceptual structures*, the *logical data structures* (those programmers are aware of) and the *physical data structures* (generally hidden, and that will be ignored in this paper).
- The *conceptual structures* of a DBMS model can be perceived as a subset of the E-R model. Indeed, such objects as *record, segment, row, tuple, entry, etc* can be considered as DBMS representation of entities. *Data items, columns* and *fields* are direct representations of entity attributes. *Set types* and *parentchild* links support the representation of binary, functional relationship types.
- The *logical data structures* of a DBMS model, making up the *logical implementation model*, comprise the conceptual structures *plus* access structures to data. Among the access structures, two are of particular importance, namely **access paths** and **files**.

The GER model defined in section 5 is a convenient medium to describe DBMS-dependent schemata at the conceptual level. It needs very few additional constructs to accomodate logical implementation specifications such as access paths and files.

ACCESS PATH
An access path is a mechanism the function of which is to gain selective (and generally quick) access to data satisfying some selection criteria. *Indices* (relational DBMS and ISAM files), *CALC keys* (DBTG DBMS), *owner/parent* and *members/children* paths (network and hierarchical DBMS), *search keys* and *indexed sets* (DBTG), are important examples of access paths in traditional DBMS.
More formally, the components of an access path are conceptual objects with which an access mechanism is associated. From the functional viewpoint, any access path mechanism falls into one of four categories according to *what* it supplies : (1) attribute values associated with an entity, (2) entities associated with given attribute values, (3) entities or attribute values of a relationship, (4) relationships comprising given entities and/or attribute values. Therefore, the specification of an access path is made up of two aggregates of GER attributes (describing E-R attributes or roles) of an entity relation schema or of a relationship relation schema. The second (or **target**) aggregate defines what can be obtained by specifying values/entities for components of the first (or **origin**) aggregate. The similarity with functional dependencies is obvious : the access path is a (mono- or multivalued) function providing target values from origin values within entities or relationships of a given type. Hence the following syntax.

Suggested syntax

access-path-decl ::= **path(**rel-designation**) :** role-attr-list $\longrightarrow$ role-attr-list
role-attr-list ::= ET-role-attr-list | RT-role-attr-list

FILE
Generally speaking, a file is a repository in which records can be stored. *Dbspace*, *tablespace*, *dataset*, *area*, *realm*, are some practical names for that concept in current DBMS.
The main logical properties of a set of files are that (1) they have distinct names and (2) they partition the set of entities (an entity is in one and only one file). Therefore, a file can be most simply formalized as some kind of basic entity domain. Due to the specific interpretation of that entity subdomain, an explicit syntax for file declaration is suggested as follows. The *file-description* clause defines a file, while the *file-contents-declaration* declares in which file(s) entities of a given type can be stored.

Suggested syntax

file-desc ::= file-name **: file**
file-contents-decl ::= Entity-set **:** file-name
Entity-set ::= ET-name | ET-algebraic-constr

DBMS-COMPLIANT GER SCHEMA

Specifying data structures at the DBMS level raises the important issue of wether a given schema complies with the DBMS model or not. That question can be solved by specifying the structural properties the schema has to satisfy in order to be recognized *DBMS-compliant*. Each DBMS can be characterized by such a set of structural rules. Rules 6.1 lists, in some natural language, some important properties every GER schema must satisfy to be DBTG-71/73- compliant (derived from [HAINAUT,86]). Rules 1 to 6 define the DBTG restrictions at the conceptual level, while rules 7 to 10 concern implementation level. Rule 1 states that specialization is not allowed; rule 2 defines *set-types* as relationship

types that are binary, N-1 and with no attributes; rule 3 prohibits *multi-ownership* (while multi-membership is allowed); rule 4 says that owner and members of a *set-type* are distinct; rule 5 states the uniqueness of *calc key* within a *record-type*; rule 6 defines hybrid identifiers within one set-type. Rule 7 states that an identifier must be an access path and rule 8 states that only one such access path is allowed. Rules 9 and 10 define set-types as two, inverse, access paths. Rule 11 states that any record must be stored into a file.

1. basic entity domains only,
2. no relationship relation schemata,
3. an entity domain can be described only once,
4. a described entity domain cannot appear more than once in a relation schema,
5. at most one all-attribute key per relation schema,
6. a key may include one role only; if so, it must include at least one attribute as well,
7. for each all-attribute key K there is an access path
   K ⟶ ET-name
8. there is at most one all-attribute access path per relation schema,
9. for any role R, there is an access path R ⟶ ET-name,
10. for each role/attribute RA of an entity relation, there is an access path ET-name ⟶ RA.
11. each entity type must be assigned to at least one file.

Rules 6.1 - Some rules for DBTG-compliant GER schemata. For simplicity, the term *attribute* means *GER attribute that represents an E-R attribute*, and the term *role* means *GER attribute that represents a E-R role*.

Rules 6.2 proposes the same exercice for conceptual structures of relational schemata. Entities correspond to t-uples the same way they correspond to records in a DBTG GER specification. A similar set of rules can be defined for hierarchical, shallow (TOTAL,IMAGE), simple files as well as for E-R DBMS.

1. no complex value domains,
2. basic entity domains only,
3. no relationship relation schemata,
4. an entity domain must be described once and only once,
5. a descriptive relation schema can include one entity attribute only,
6. the only constraints are keys defined on non-entity attributes

Rules 6.2 - *Main rules for SQL-compliant GER schemata.*

Example 6.3 illustrates the use of extended GER concepts for the specification of a DBTG-compliant data base schema. The first section declares the conceptual structures expressed into record-types CUSTOMER and ORDER, and the set-type SENDS, represented in the concise version.The second section defines two files and the relationships with entity types (note that CUSTOMER entities are distributed according to their ADDRESS value). The third section defines the access paths. For instance, (2) and (4) specify the DBTG CALC keys, while (5) and (6) specify owner/member and member/owner access through sets SENDS; (7) can be the formalization of an indexed set. *Note*. Access-paths (1),(3),(5) and (6) are automatically provided by the DBMS as specified in Rules 6.1 hereabove.

Therefore, only specifications (2),(4) and (7) need be declared in the context of a DBTG DBMS.

```
CUSTOMER : ENTITIES
ORDER : ENTITIES
desc-of-CUSTOMER(CUSTOMER,CUSTNUM,CUSTNAME,ADDRESS)
desc-of-ORDER(ORDER,ORDNUM,DATE,SENDS: CUSTOMER)

FCUSTL : file
FCUSTX : file
desc-of-CUSTOMER(ADDRESS = 'London')[CUSTOMER] : FCUSTL
desc-of-CUSTOMER(ADDRESS ≠ 'London')[CUSTOMER] : FCUSTX
ORDER : FCUSTL
ORDER : FCUSTX

path(desc-of-CUSTOMER) : CUSTOMER ⟶ CUSTNUM,CUSTNAME,ADDRESS  (1)
path(desc-of-CUSTOMER) : CUSTNUM  ⟶ CUSTOMER              (2)
path(desc-of-ORDER) : ORDER ⟶ ORDNUM,DATE                 (3)
path(desc-of-ORDER) : ORDNUM ⟶ ORDER                      (4)
path(desc-of-ORDER) : SENDS ⟶ ORDER                       (5)
path(desc-of-ORDER) : ORDER ⟶ SENDS                       (6)
path(desc-of-ORDER) : DATE,SENDS ⟶ ORDER                  (7)
```

Example 6.3 - *GER description of a DBTG-compliant logical implementation schema.*

7. SCHEMA TRANSFORMATION IN THE GER MODEL

Proposing a generic framework for a wide class of existing models is bound to be an academic essay, and therefore practically useless, if that framework is not associated with translation or transformation rules between models and between schemata expressed in these models. Building such rules benefits considerably from the relational expression of the GER model. Indeed, relational algebra is known to be a sound and complete fondation for first order data derivation. Moreover, current extensions to relational algebra allow deeper structural modifications needed in N1NF models [JAESCHKE,82], [ABITEBOUL,87]. On the other hand, the model translation problem (i.e. finding general rules for translating any schema in model M1 into an equivalent schema in model M2) is reduced to the simpler problem of schema translation within a unique model, namely the GER model. This is particularly fruitful when binary models are concerned [ABRIAL,74], [VERHEIJEN,82], since the latter are known not to be complete [DATE,85] as far as operator application and constraint expression are concerned.

Space is lacking in this paper to deal with the problem of model and schema transformation in sufficient detail. A set of schema transformation rules valid for a large subset of the GER model were proposed in [HAINAUT,88] and [HAINAUT,89]. These rules are proved to be reversible, i.e. they induce no loss of semantics. Moreover, reversible transformations were proposed concerning the access path structures of GER binary schemata [HAINAUT,86].

8. CONCLUSION

The paper has laid down the basic principles of a fairly comprehensive, non redundant and rigorous set of concepts that cover the main data models currently used today in the realm of Information Systems design. Starting from an extended relational model, we have analysed the main constructs currently proposed in the E-R model family. We have successfully tried to give a natural relational translation of these constructs. We have then reduced the extended relational model in such a way that (1) each basic and extended E-R construct has a unique relational expression and (2) each relational construct has a unique E-R interpretation, leading to the equivalence of both models. The resulting set of concepts, the GER model, appears to be simpler and more general than current E-R models. However, it is not intended to be one more proposal for data base schema specification. Its main objectives are to offer a general framework for analysing, comparing and translating competing models, and to provide a unique, consistent specification context for data base design from the conceptual level to the DBMS-dependent level.

The main ideas which underlie the GER model are the relational interpretation of the E-R concepts, the explicit representation of entities with non-property domains, and the use of complex domain obtained by powerful constructors. None of these ideas is new, but their merging leads to a unique modeling universe of concepts that elegantly encompass those of most current models of the E-R, relational, binary and N1NF families.

That general framework has already led to the building of a semantics- preserving transformational toolset for model conversion and for schema transformation in the context of Information System design [HAINAUT,81,-86,-88,-89]. Moreover, a family of DB-oriented CASE, based on the concept of a unique E-R model for all the design layers, and on the transformational approach have been designed and marketed [CADELLI,87] [HAINAUT,87].

REFERENCES

ABITEBOUL,87, ABITEBOUL, BEERI, "On the power of languages for the manipulation of complex objects", INRIA technical report, 1987

ABRIAL,74, ABRIAL, "Data Semantics", in *Data base management*, North-Holland, 1974

BACHMAN,69, BACHMAN, "Data Structure Diagrams", Data Base, 1, N°1, ACM SIG on Business Data Processing, 1969

BODART,88, BODART, PIGNEUR, "Conception assistée des applications informatiques - 1. Etude d'opportunité et analyse conceptuelle", MASSON, Paris 1988

BRACCHI,76, BRACCHI, PAOLINI, PELEGATTI, "Binary logical associations in data modelling", in *Modelling in data base management systems*, North-Holland, 1976

CADELLI,87, CADELLI, CHARLOT, DELCOURT, HAINAUT, "L'atelier de conception de base de données ORGA", Technical report, Institut d'Informatique, FUNDP, Namur, April 1987

CHEN,76, CHEN, "The entity-relationship model - toward a unified view of data", ACM TODS, Vol. 1, N° 1, 1976

CODASYL,62
Development Committee : An information algebra, Com. ACM, 5, n° 4, April 1962

CODD,70, CODD, "A relational model for large, shared data banks", Commun. ACM Vol. 13, N° 6, 1970

CODD,79, CODD, "Extending the Database Relational Model to capture more meaning", ACM TODS, Vol.4, N°4, 1979

COLLART,88, COLLART, JORIS, "Etude théorique et pratique d'extensions au modèle Entité-Association", Master Thesis, Institut d'Informatique, FUNDP, Namur, 1988 (french)

DATE,85, DATE, "An introduction to database systems", Vol II, Addison-Wesley, 1985

DATE,86, DATE, "An introduction to database systems", Vol I, Addison-Wesley, 1986

DATE,86b, DATE, "Relational Databases - Selected writings", Addison-Wesley, 1986

ELMASRI,85, ELMASRI,WEELDREYER,HEVNER, "The caregory concept : an extension to the entity-relationship model", Data and Knowledge Engineering, Vol.1, n°1, 1985

FIKES,85, FIKES, KEHLER, "The role of frame-based representation in reasoning", CACM, Vol. 28, n° 9, Sept. 1985

GYSSEN,88, GYSSEN, VAN GUCHT, "The powerset algebra as a result of adding programming constructs to the nested relational Algebra", Proc. SIGMOD Conf., Chicago, 1988

HAINAUT,74, HAINAUT, LE CHARLIER, "An extensible semantic model of data base and its data manipulation language", Proc. IFIP Congress, North-Holland, 1974

HAINAUT,81, HAINAUT, "Theoretical and practical tools for data base design", in Proc. Intern. VLDB conf., ACM/IEEE, 1981

HAINAUT,86, HAINAUT, "Conception assistée des applications informatiques - 2. Conception de la base de données", MASSON, Paris 1986

HAINAUT,87, HAINAUT, CADELLI, CHARLOT, DELCOURT, "Conception de bases de données - Eléments méthodologiques", Research report, Institut d'Informatique, FUNDP, Namur, March 1987

HAINAUT,88, HAINAUT, "Non conventional transformations of data base schemata", Draft Research report, Institut d'Informatique, FUNDP (in french), Dec. 1988

HAINAUT,89, HAINAUT, "Semantic equivalence of Data Base Schemata - A new family of formal tools", Research Report, Institut d'Informatique, FUNDP, June 1989 (submitted for public.)

HALL,76, HALL, OWLETT, TODD, "Relations and Entities", in *Modelling in Data Base Systems*, North-Holland, 1976

HAMMER,81, HAMMER,MCLEOD, "Database description with with SDM : a semantic database model", ACM TODS, Vol. 6, n°3, 1981

HULL,87, HULL, "A survey of theoretical research on typed complex database objects", in *International Lecture Series in Computer Science*, Academic Press, 1987

JUNET,87, JUNET, "Design and implementation of an extended entity-relationship database management system", in Proc. of the 5th Conf. on E-R approach, North-Holland, 1987

KOZACZYNSKY,87, KOZACZYNSKY, LILIEN, "An extended Entity-Relationship (E^2R) database specification and its automatic verification and transformation", Proc. 6th Entity-Relationship Conf.,1987

MAIER,83, MAIER, "The Theory of Relational Databases", Computer Science Press, 1983

MAKINOUCHI,77, MAKINOUCHI, "A consideration of normal form of Not-Necessarily- Normalized Relations in the Relational Data Model", Proc. 3rd VLDB Conf., Tokio, 1977

MEES,87, MEES, PUT, "Extending a dynamic modelling method using data modelling capabilities : the case of JSD", in Proc. of the 5th Conf. on E-R approach, North-Holland, 1987

MOTRO,87, MOTRO, "Superviews: Virtual Integration of Multiple Databases", IEEE Trans. on Soft. Engin. Vol. SE-13, N°7, July 1987

PARENT,86, PARENT, SPACCAPITIETRA, "Enhancing the operational semantics of the entity-relationship model", Proc. of IFIP WG 2.6 WC on Data Semantics, North-Holland, 1986

PECKHAM,88, PECKHAM, MARYANSKI, "Semantic Data Models", ACM Comp. Surveys, vol.20,n°3, 1988

SCHEUERMANN,80, SCHEUERMANN,SCHIFFNER,WEBER, "Abstraction capabilities and invariant properties modelling within the E/R approach", in Proc. of the 1st Conf. on E-R approach, North-Holland, 1980

SMITH,77, SMITH, SMITH, "Database abstractions : aggregation and generalization", ACM TODS, Vol. 2, N° 2, 1977

TEOREY,86, TEOREY,YANG,FRY, "A logical design methodology for relational databases using the Extended Entity-Relationship model", ACM Computing Surveys, Vol. 18, N° 2, June 1986.

SHIPMAN,81, SHIPMAN, "The functional data model and the data language DAPLEX", ACM TODS, Vol. 6, N° 1, 1981

ULLMAN,88, ULLMAN, "Principles of database and knowledge-base systems", Computer Science Press, 1988

VERHEIJEN,82, VERHEIJEN, Van BEKKUM, "NIAM : An information system analysis method", in *Information Systems design methodologies*, North-Holland, 1982

WOODS,75, WOODS, "What's in a link : foundation for semantic networks", Representation and Understanding, Academic Press, 1975

YAESCHKE,82, YAESCHKE, SCHEK, "Remarks on the Algebras on Non First Normal Form Relation", Proc. 1st PODS, Los Angeles, 1982

APPENDIX A : a definition language for the extended relational model

The following definitions outline a specification language for both the standard relational structures and the recent extensions. The meta-syntax is as usual. A boldface **word** denotes a terminal symbol and italic *sentences* are used as informal shortcuts.

Schema declaration

schema-decl ::= declarations relation-decl declarations
declarations ::= *possibly empty list of :* subdomain-decl | relation-decl | constraint-decl

Domain declaration

subdomain-decl ::= subdomain-name **:** domain-designation
domain-designation ::= domain-name | domain-constructor
domain-name ::= subdomain-name | basic-domain-name | relation-name
basic-domain-name ::= **integer** | **real** | **char(**integer**)** | **date** | **ENTITIES** | *etc*
domain-constr ::= cartesian-constr | powerset-constr | algebraic-constr | list-constr
cartesian-constr ::= **(** attribute-def-list **)**
attribute-def-list ::= attribute-def | attribute-def **,** attribute-def-list
attribute-def ::= attribute-name **:** domain-designation | subdomain-name
powerset-constr ::= subdomain-name **[** size-range **]**
size-range ::= integer | integer**:**integer | ***** | integer**:***

algebraic-constr ::= union | intersection | difference | join | select | project |
aggregate | collapse | *etc*
list-constr ::= **{**value-list**}** | **{}**
value-list ::= value | value **,** value-list

Relation declaration (optional)

relation-decl ::= relation-name cartesian-constr

Constraint declaration

constraint-decl ::= key-decl | dependency-decl | inclusion-decl | *etc*

key-decl ::= **key(**relation-designation**) =** attribute-list
relation-designation ::= relation-name | algebraic-constructor *resulting in a relation*
attribute-list ::= attr-designation | attr-designation **,** attribute-list
attr-designation ::= attr-id | attr-id**.**attr-designation
attr-id ::= attribute-name | attribute-name**[*]**

dependency-decl ::= rel-designation **:** attr-name-list dependency-op attr-name-list
dependency-op ::= $\longrightarrow$ **|** $\rightarrow\!\!\!\!\rightarrow$

inclusion-decl ::= domain-designation inclusion-op domain-designation
inclusion-op ::= $\subseteq$ | **=**

relation-name,attribute-name,integer and subdomain-name : are character strings
union,intersection,difference,join,select,project are set and relational operators of relational algebra
aggregate, collapse, telescope, nest, unnest, etc, are operators for structural reorganization of set and/or cartesian constructed attributes [MOTRO,88], [HULL,87]
value is any constant representation