• 1400014058
Còpia 4

**A Methodology for Semantically Enriching
Interoperable Databases**

Malú Castellanos

Report LSI-93-1-R

UPC
Facultat d'Informàtica
de Barcelona - Biblioteca
- 2 MARÇ 1993

A Methodology for Semantically Enriching Interoperable Databases

Malú Castellanos

Dept. de Llenguatges i Sistemes Informàtics

Universitat Politècnica de Catalunya, Spain

castellanos@lsi.upc.es

Abstract

Integrated access to a Federated Database System, whatever its architecture is, requires a deep knowledge about the semantics of its component databases so that interdatabase semantic relationships can be detected. Unfortunately, very often there is a lack of such a knowledge and the local schemas, being semantically poor as a consequence of the limited expressiveness of traditional data models, do not help to acquire this knowledge. The solution to overcome this limitation is to upgrade the semantic level of the local schemas thru a semantic enrichment process that discovers implicit knowledge and makes it explicit by converting the local schemas to rich schemas expressed in a canonical model. Here we present a methodology for semantic enrichment describing its two phases, knowledge acquisition where restrictions in the form of identifiers of various types and dependencies of different kinds are discovered by analysing the intension as well as the extension of the database, and conversion of schemas where the original metadata augmented with the discovered one are converted to rich schemas. The relevant implications of this process for the integrated access is pointed out.

1. Introduction

In Federated Database Systems (FDBS) [SheLar90] autonomous databases cooperate sharing their data while at the same time maintaining their autonomy. The user is provided with an interface that hides the syntactic and semantic heterogeneities that exist among the component databases in order to access the FDBS in an integrated way. Syntactic heterogeneities, consequence of the use of different data models, are commonly solved by the use of a *canonical* model to which the local schemas are converted. However, even if all schemas are expressed in the same data model, a same real world concept may be represented in different ways in the component databases, leading to *semantic heterogeneities* that must be identified for accessing the FDBS in an integrated way. Moreover, not only equivalent concepts must be detected, but also any kind of semantic relationship among the concepts of the databases that constitute the federation.

The detection of interdatabase semantic relationships requires a deep knowledge about the meaning of the databases. However, there is often a lack of such a knowledge, specially when there are many different independent databases as in the interoperability context. This problem is augmented by the fact that the database schemas do not help to acquire this knowledge because their semantic content is very poor due to the limited expressiveness of the traditional data models in which they are expressed. A solution to overcome this limitation of the schemas is to upgrade their semantic level thru a *semantic enrichment* process where implicit knowledge is discovered and made explicit by converting the schemas to a rich canonical model. This corresponds to the two phases of our methodology described in this paper. In the *knowledge acquisition* phase, implicit knowledge in the form of restrictions, like keys and different kinds of dependencies, is extracted by analyzing the schemas and the extensions of the databases. Since analyzing the extension of the database is a costly operation, emphasis has been in developing algorithms that eliminate redundant work. The knowledge acquired in this phase is made explicit in the second phase, called *schema conversion*, where the local relational schemas are converted to our rich canonical BLOOM model. The methodology constitutes the basis of a tool that semiautomatizes the enrichment process.

In the first works on integration of databases [BatLenNav86] it was assumed that the semantic relationships between the component databases had already been detected by some means, so that they only considered how to merge the database schemas according to these relationships. However, it is clear that an integration methodology should deal with both aspects of the problem; moreover, it is the specification of these relationships the most critical task in the integration of schemas. Semantic heterogeneity [She91] that has arisen as an area of big interest recently, complicates enormously this detection task where various aspects of the component schemas are

analysed and compared in order to identify the semantic relationships that exist between them. In order to decrease the complexity of this task and to provide a semi-automatic support for it, mechanisms to guide the search process in a systematic way and to reduce the number of comparisons must be developed. One such mechanism presented in [GarCas92] is based on the rich abstractions of the BLOOM schemas resulting from the enrichment process

This paper is organized as follows. In section 2 related work is discussed. The BLOOM model is outlined in section 3. Section 4 discusses the knowledge acquisition phase of the enrichment methodology while the conversion of schemas is presented in section 5. Our conclusions and future work are presented in section 7

2. Related Work.

The subject of mapping a DB schema from one model to another has been studied by many authors. Most of the work refers to mappings from a more semantic model to a less semantic one, particularly from a conceptual schema in the Entity-Relationship (ER) model or some extensions of it, to a DBMS schema expressed in the relational model. However, the subject of mapping from a less semantic model to a more semantic one, as is needed for interoperability, has been less studied. In fact, it is more than just a mapping, since knowledge must be acquired in some way in order to upgrade the schema to a higher semantic level. As far as we know all the work on conversion to a richer model use the relational model as the source data model and the ER model or some extension of it, as the target one. A brief overview of this work is presented in [CasSal91]. Furthermore, in some of them, the ER schema is just a graphical means of representing the relational schema ([CasAma83]), while in others where the semantic contents is really enhanced, they either rely entirely on the DBA to obtain all the necessary knowledge interactively [NavAwo87], or assume that it has been specified in advance ([JohKal89], [MarMak90]). Besides, some of them consider only partial implicit knowledge, by considering only primary keys ([DavAro87]), or by making some candidate key substitutions ([NavAwo87]), or consider only well designed schemas and consider only key-based inclusion dependencies ([MarMak90]).

In contrast, our approach is general enough to cover both, well and bad designed schemas, as required for the interoperability context where the pre-existing databases could have been designed in any way. We consider every single candidate key as well as every single inclusion dependency, either key and non key based, also exclusion and complementariness dependencies are taken into account. Moreover, we include a knowledge acquisition phase to discover as much implicit semantics as possible by analysing the schema and extension of the database, and only require minimal user intervention when ambiguities arise and to confirm results.

The work presented in this paper also differs from other work in that we do not use an ER model as the target one, instead we use the BLOOM model which is a semantic extension of an object oriented model because it complies with all the characteristics of suitability in expressiveness and semantic relativism of canonical models reported in [SalCasGar91]. ER models does not satisfy them, in particular, its dichotomy (entity versus relationships) creates problems both in the enrichment and in the integration processes.

Our method is more global and general and the resulting schemas are richer than in the other works. Finally, our approach is oriented towards the interoperability of databases, although it is applicable in other contexts.

As stated above, our method includes a knowledge acquisition phase where keys and dependencies are inferred. Again our approach is general enough to cover the case where they are not supported by the DBMS, as happens in practice for most of the dependencies, and they must be obtained from the extension of the database. As far as we know, the only work on dependency inference that analyses the extensions of the relations is the work reported in [ManRai88] where functional dependencies are obtained

3. The BLOOM Model.

The choice of the canonical model used for the federated database system is critical in the process of schema integration. It must comply with several characteristics of expressiveness and semantic relativism reported in [SalCasGar91] that make it adequate for this process. In particular it must be rich enough to model the semantics already expressed in the local schemas, as well, as the semantics obtained from a semantic enrichment process to upgrade the semantic level of the schemas (section 4) so that similarities among classes

can be detected easily and interdatabase semantic relationships can be specified. With this purpose in mind we have designed BLOOM.

BLOOM is an extension of an object oriented data model whose main constituents are objects and classes. Classes describe the structure and behaviour of their object members. The description of the structure entails the description of properties and relationships to other object classes. Modeling with BLOOM allows to capture a rich set of semantic relationships which are characterized by the abstractions that they represent:

- Classification/Instantiation

- Four kinds of Generalization/Specialization:

Disjoint Specialization: each object of the superclass belongs *at most* to one subclass.

Complementary Specialization: each object of the superclass belongs *at least to* one subclass.

Alternative Specialization: each object belongs to *one and only one* subclass.

General Specialization: this one has no restrictions.

Three kinds of Aggregation:

Simple Aggregation: this is the simplest type of aggregation and is employed to express that the attributes involved in it are just properties of the object being described and nothing else. Each attribute takes as domain either a primitive class such as integer, or a user defined class such as department. We distinguish this by the concepts *primitive* and *derived* respectively. Attributes can be single valued or multi valued, single valued is the default while multivalued attributes use the *set_of*. Attributes can be *obligatory*, if they don't accept nulls; by default they accept nulls. The construct for this abstraction is *s_aggr_of*, and its inverse *s_partic_in*.

Collection Aggregation (or just *collection*): the collection of some objects of a given class gives rise to a new complex object. This abstraction permits to treat each collection of objects as a single one. This concept introduced in [HamMcL78] corresponds to the 'cover aggregation' in [Cod79] The construct for it is *collection_of* and its inverse *collected_in*.

Association Aggregation (or just *association*): the aggregate object is formed by aggregating objects from different classes. The component objects are not simply properties of the aggregate but it is their association what gives rise to it. The construct is *associates* and its inverse *associated_in*. The aggregate object, in turn, has its own properties specified by the simple aggregation.

- Two kinds of specific dependencies (besides those existence dependencies inherent to the other abstractions) to model two different semantic situations:

Interest dependency: an object is of interest only as long as another object on which it depends is of interest, like the children of the employees. The construct for it is *int_dep_on* and its inverse *has_int_deps*.

Existence dependency: an object cannot be created if the object on which it depends does not exist, but once created its existence does not depend any more on the other one. For example orders can only be placed to authorized suppliers, but if an authorization is deleted, the orders that have been put on it are maintained for historic record

The operational functionality of BLOOM models the dynamic properties which can range from the simple specification of insertion and deletion constraints to the modeling of operations. The insertion and deletion constraints used to maintain the integrity of the semantic database constitutes the behavioural interpretation of the semantics of the model which is specified in the metaclasses corresponding to the different abstractions [CasSalGar91]. Metaclasses in BLOOM do not only specify the behaviour of their class instances, but also that of the instances of their instances. Furthermore, metaclasses permit to adapt the model to different contexts by extending it with other abstractions.

BLOOM has a whole set of operators for integration, in particular, discriminated operations like discriminated generalization and specialization by attribute useful for overcoming schematic discrepancies [SalCasGar92] and for building federated schemas that support multiple semantics [GarSal91].

4. Knowledge Acquisition.

The knowledge acquisition phase takes as input the relational database, the metadatabase that stores its schema, and some information requested to the user when ambiguities arise. In each step of this phase a different aspect of this information is analysed in order to extract some kind of semantic knowledge, and the metadatabase is augmented with it as shown in the next figure.

Since the purpose of the enrichment process is to obtain rich descriptions of the objects, and the semantics of an object is given in big extent by its semantic relationships to the other ones, the focus of our approach is on discovering the nature of these relationships. The implicit or explicit restrictions imposed to the database in the form of keys and dependencies embody this kind of semantic information, thus, the goal of the knowledge acquisition phase is precisely to discover them thru an analysis of the extensions and the schemas of the databases.

Relational DBMSs provide different degrees of support (definition and enforcement) for the different kinds of restrictions. *Direct* support exists as a consequence of special purpose data definition language (DDL) statements to define specific types of restrictions in a straightforward way, while in the *indirect* support these specific statements do not exist but more general ones can be used indirectly for this purpose. In both cases the definitions will be stored as metadata in the catalog. When no DDL statements can be used in any way to define restrictions, we say that there is *no support*. In this case there is no information about restrictions in the catalog and the only way to discover them is by analysing the database extension.

Step 1: Key Inference.

In this step we obtain all the keys (simple and composite) for each relation. It is not necessary to obtain their type, primary or secondary, because as will be seen in 2.4 design anomalies covered in our methodology make irrelevant this distinction. Keys can be inferred in various ways depending on the underlying information.

- a) **Schema Support:** keys are specified in some way when the schema is defined, thus, the corresponding information can be retrieved from the metadata in the catalog.
 - a.1) **Direct Support:** in some relational DBMSs (RDBMS) primary keys (PKs) are specified thru the special statement 'primary key', when the relation is being defined, so that the system can enforce entity integrity. Secondary keys cannot be defined directly, so they have to be inferred by a.2) or b).
 - a.2) **Indirect Support:** in any RDBMS it is possible to specify keys indirectly by means of the 'unique index' or 'unique' constraint in combination with the 'not null' constraint, so that the system can enforce the two constraints that a key must comply.
- b) **No Support:** even that in any RDBMS there is at least indirect support for keys, we have considered the possibility that some secondary key has not been specified. Thus, an algorithm for inferring keys from the extensions of the relations has been developed [Cas92]. Different combinations of attributes are checked for the uniqueness property by comparing its cardinality with the one of the relation, but since this is a costly operation, redundant work is avoided by obtaining only minimal keys.

Step 2: Functional Dependency Inference.

Our algorithm for discovering functional dependencies (FDs) of the type lhs $\rightarrow$ rhs (left hand side functionally determines right hand side) from the extension of the DB, establishes an order in the sequence of left hand sides to be tested. Since we must test that tuples with the same value for 'lhs' also agree in the value for 'rhs', we sort the relation by using 'lhs' as the sort key. Furthermore, the candidate lhs's are grouped by the maximal sort key in order to reduce the number of sorts to be performed (for example, a sort by ABCD is used to test the candidate lhs's A, AB, ABC, and ABCD). For every pair of consecutive tuples that are compared, all the lhs's which use the same sort are tested, and for each lhs, all its possible rhs's are compared. The idea is that once two tuples have been retrieved, we test all that we can test on them. Also, once a FD is discovered, we apply the transitivity rule to obtain derived dependencies and for every FD to be proposed as candidate, we first check if it is not redundant by the augmentation rule. In this way, our algorithm developed in [3] minimizes the number of disk accesses.

Step 3: Normalization.

The importance of normalizing the relations for the enrichment process is to obtain relations that describe a unique concept. Furthermore, by normalizing to third normal form, the semantics of the functional dependencies will not be lost. Also, since the classes resulting from the enrichment process will describe unique concepts, this helps in the detection of interdatabase relationships. Any standard algorithm for normalization can be used. The input for this process is the set of FDs and the output is the set of normalized relations along with their keys.

Step 4: Determination of the Type of the Identifiers.

One of the goals of our methodology is to encompass an as large as possible class of relational schemas, so it doesn't rely on the assumption that the schemas are well designed. Anomalies like making foreign keys match secondary keys, specially if the secondary key refers to a missing entity (just modelled as an attribute), or choosing as primary key one candidate key that does not correspond to the intention of the relation, are considered in our methodology. These anomalies make unnecessary to obtain the type of the key when it is discovered. We need a mechanism which adds a more precise semantics to all candidate keys. For this purpose, we introduce two types of identifiers:

- a) **Proper identifier** (P_{id}) :- this is a key whose identification role results adequate according to the intension of the relation.
- b) **Extraneous identifier** (E_{id}) :- it is a key whose identification role is just of syntactic nature, that is, even though it complies with the uniqueness and minimality properties, it is not an adequate identifier from a semantic point of view because it does not correspond to the intension of the relation.

Example: department (dept_no, d_name, manager, budget, location)
 where: dept_no → proper identifier
 d_name → proper identifier
 manager → extraneous identifier: given a manager the corresponding department is determined but the relation describes departments and not managers

Appropriate questions are posed to the user for obtaining the information that the system requires to determine the type of identifier corresponding to each key.

Step 5: Inclusion Dependency Inference.

Inclusion dependencies (INDs) constitute the basis of the conversion phase, because from their analysis, the nature of the relationships among the objects of the database can be determined. Since the methodology doesn't rely on any convenient well behaviour of schemas, an attribute in a relation R' can reference any kind of attribute in another relation R and not only keys. Thus, INDs in all generality have to be obtained.

If normalization has been applied, the obtained INDs do not correspond to the new schema, but since normalization preserves dependencies, they can easily be mapped to it by keeping record of the attributes that

were projected out to new relations. Moreover, each decomposition performed in the normalization step gives rise to a new key_based IND and its inverse one between the attribute projected as the key of the new relation and the same attribute kept in the original relation as foreign key (join attributes). Analogously to the extraction of keys, there are different kinds of support for INDs:

a) **Schema Support:** DDL statements are provided to specify INDs and these definitions are kept in the catalog.

a.1) **Direct Support:** key based INDs are supported as foreign keys when the DBMS supports the referential integrity constraint.

a.2) **Indirect Support:** if the system supports user defined constraints where the conditional part may refer to another table, then, INDs in general can be specified thru this mechanism (from the analyzed RDBMSs only the 'check' constraint of Rdb 3.0 provides this capability).

b) **No Support:** there is no mechanism to define INDs in the schema so they have to be inferred by analyzing the extensions of the relations. Emphasis must be put on the efficiency of the inference process. The naive method of checking all possible INDs, one by one, is practically impossible for two reasons: one is that for any pair of relations there are more than $n!$ possible non equivalent IND, and the second one is that in general it is not possible to check the existence of a long IND fast (NP complete). However, we can use some heuristics (A to E) to reduce the set of possible INDs, and to reduce the number of comparisons for the remaining possible INDs (F & G):

A. *The corresponding attributes in the left and right hand side of an IND must be of the same domain or at least of the same type.*

B. *Only those domains/types used for identifying purposes are considered.*

Since INDs constitute the relational mechanism for referencing entities thru their identifiers, only syntactic domains used for identification purposes have to be considered, that is, strings of characters and numbers without decimal part.

C. *Start with unary INDs.*

D. *Disregard all those possible non unary INDs for which there does not exist a unary IND for each pair of the corresponding attributes.*

The last two heuristics are based on the projection axiom of INDs:

if there is an IND: $R [a_1, a_2, \dots, a_n] \subset S [b_1, b_2, \dots, b_n]$

then the INDs: $R [a_1] \subset S [b_1], R [a_2] \subset S [b_2], \dots, R [a_n] \subset S [b_n]$ exist.

Thus, unary INDs constitute necessary conditions for the existence of non unary ones.

D. *A limit must be imposed on the length of INDs considered.*

Typically the INDs that hold in a database are short, so it is not useful to consider INDs of arbitrary length. A limit of three seems rather reasonable.

E. *Compare an attribute only with those of the same type with greater of equal cardinality.*

It makes no sense to look for an IND where the attribute on the right hand side (rhs) of the IND has lower cardinality than the one on the left hand side (lhs), since the values of this last one are not a subset (neither proper or improper) of the other one. For this purpose (as well as for the ones of other heuristics) it is very important that the list of attributes of the same type be ordered according to the cardinalities of the attributes.

F. *Never compare an attribute A_i with another one A_j with greater (or equal) cardinality before the other attributes with greater cardinality have been compared with A_j .*

The comparison of attributes is a costly operation because it requires to project them and perform a difference in order to check whether the extension of one of them is contained in the extension of the other. To minimize the comparison work transitive INDs are *always* obtained thru the simple application of the transitivity axiom. Thus, a possible IND $A_i \subset A_j$ is considered in the comparison process only when we can guarantee that it is not a transitive dependency. In order to follow this heuristic we have made the following considerations for the algorithm reported in [Cas92]:

- attributes of the list are considered to constitute the lhs of possible INDs one by one in *descending* order according to their cardinalities (outer loop),
- an attribute A_i on the lhs of searched INDs is compared with those (possible rhs) of the same type in *ascending* order according to their cardinalities (inner loop),
- as soon as an IND is found, immediately compute all transitive dependencies of A_i . This can be done since the INDs for attributes with higher cardinalities have already been computed,
- when an IND $A_j \subset A_i$ is found, check if the inverse IND $A_i \subset A_j$ exists, if so, then both attributes are the same and A_j has exactly the same INDs, previously obtained, as A_i .

The next example illustrates the procedure to discover INDs:

Step 6: Exclusion and Complementariness Dependencies.

These two types of dependencies are used in the next phase of the methodology in order to determine the kind of specialization (alternative, complementary, disjoint or general) when there is such a relationship.

An exclusion dependency (EXD) exists when two or more (subsets of) attributes are mutually exclusive, while in a complementary dependency (CD) the union of them gives the extension of the parent.

a) **Indirect Schema Support:** there is no special statement to define these dependencies but they can be specified indirectly in the schema as user defined constraints. Thus, they can be extracted by analyzing the appropriate information in the catalog.

b) **No Support:** nothing about the dependencies can be specified in the schema, thus, the only way to infer them is from analyzing the extensions of the relations. Fortunately, the number of comparisons to be performed can be reduced to a big extent by the following heuristic:

1. Only key attributes for which a key based IND with respect to a common key attribute exists are to be compared.

These dependencies are only used to determine the kind of specialization (disjoint, alternative, complementary or general), and since a specialization relationship between two relations can exist only when there is an IND between their key attributes; only those key attributes of the relations which have the same parent have to be compared. However, we must test the complementariness of every possible subset with respect to the parent relation and the disjointness among the members of the subset, and even when they are discovered, we do not know which of them correspond to a specialization criteria, except for the alternative specialization where a CD and an EXD exist for the relations. In this case, the sum of the cardinalities of the relations is equal to the cardinality of the parent, and the intersection between them is the empty set.

Once all the steps of the knowledge acquisition phase have been performed, the knowledge extracted in the form of keys and dependencies is prepared for the next phase by organizing it in appropriate data structures. The tasks of this first phase are performed almost automatically, requiring the intervention of the user only to provide the requested information relevant for the determination of the type of the identifiers, and for validating the results obtained.

5. Schema Conversion

Inclusion dependencies (IND's) constitute the core of the conversion phase because they can be interpreted in terms of the general abstraction principles of semantic data models, in particular, of the BLOOM model. Semantic structural relationships conform a network of connexions among the classes and it is the nature of these connexions what must be precisely defined so as to reflect all the semantics that they embody. In this way, the description of a class embodying the description of *all its interclass connexions* is much richer than its original relational counterpart.

As stated before, one objective of our methodology is to encompass an as large as possible class of relational schemas, that is, we do not assume that schemas have been well designed. In particular, an attribute can reference any kind of attribute in another table, thus, not only key based INDs are considered, but also any other kind, that is, INDs in its whole generality, $R.a \subset S.b$, as shown in the next table:

R.a \ S.b	Simple Key (SK)	Composite Key (CK)	Simple Part of CK (s.p.CK)	Composite Part of CK (c.p.CK)	Non Key (s.NK)	Non Key (c.NK)
Simple Key (SK)	S - S	---	S - s.p.C	S - c.p.C	S - s.N	S - c.N
Composite Key (SK)	---	C - C	---	C - c.p.C	C - s.N	C - c.N
Each Part of CK (e.p.CK)	e.p.C - S	e.p.C - C	e.p.C - s.p.C	e.p.C - c.p.C	e.p.C-s.N	e.p.C-c.N
Only Part of CK (o.p.CK)	o.p.C - S	o.p.C - C	o.p.C - s.p.C	o.p.C - c.p.C	o.p.C-s.N	o.p.C-c.N
Non Key (NK)	N - S	N - C	N - s.p.C	N - c.p.C	N - s.N	N - c.N

Table 1. Case Analysis of an IND $R.a \subset S.b$.

Each case of the table has in turn other subcases and all of them are treated thru the different steps of the conversion phase. The main general steps are shown in the next figure:

The main steps are those in charge of analyzing the different cases of INDs of table 1. Due to space limitations it is not possible to analyze here every case of the table, instead, only a brief discussion of the main

steps of the conversion phase will be presented and illustrated by examples where we can contrast the relational schema with the BLOOM schema resulting from the conversion. Only partial descriptions of the classes are given in the examples because the purpose is illustrating only the semantic abstractions derived from the IND being considered. In practice the system will ask the user the detail of description that he/she wants to see, that is, the level of depth in the generalization and in the aggregation hierarchies that he has interest in being displayed for a given class as shown in the next examples. In the first one, the classes are presented at depth level 1, while in the second example, we assume that the user is interested only on class 'employee' at depth level 2. The depth level in each dimension is independent from the other one.

Relational Schema:

person (ss_num, car_num, name, phone, club)
 employee (ss_num, salary, hired, job, dept)
 department (name, manager, budget)
 children (emp_name, birth, gender)
 project_assignment (emp_project, hours, delivery)
 car (plate, model, owner)
 club (name, meet_place, founded)

BLOOM Schema:

```
class person
  compl_generaliz_of
    employee, student
  collected_in club
  s_particip_in car [as owner]
  s_aggreg_of
    name: char[15]
    phone: char[8]
    id ss_num: char[10]
end_class
```

```
class children
  id name int_dep_on employee
  s_aggreg_of
    birth: date
    gender: char[3]
end_class
```

```
class car
  s_aggreg_of
    person [as owner]
    plate: char[7]
    model: char[8]
end_class
```

```
class employee
  compl_specializ_of person
  generaliz_of manager
  has_int_deps children
  s_set_particip_in department
  associated_in proj_assignment
  s_aggreg_of
    salary: float
    hired: date
    job: char[8]
end_class
```

```
class project
  associated_in
    proj_assignment
    s_aggreg_of
      p_name: char[8]
end_class
```

```
class manager
  specializ_of employee
  s_particip_in department
end_class
```

```
class department
  s_aggreg_of
    set_of employee
    manager
    budget: float
    id name: char[10]
end_class
```

```
class proj_assignment
  association_of
    employee
    project
  s_aggreg_of
    hours: float
    delivery: date
end_class
```

```
class club
  collection_of person
  s_aggreg_of
    meet_place: char[15]
    founded: date
    name: char[12]
end_class
```

Example 1. Depth level 1 in both dimensions

```

class employee
  compl_specializ_of person
  generaliz_of manager
  collected_in club {inherited}
    s_aggreg_of
      meet_place: char[15]
      founded: date
      name: char[12]
  has_int_deps children
    s_aggreg_of
      birth: date
      gender: char[3]
      name: char[15]
  s_set_particip_in department
    s_aggreg_of
      manager
      budget: float
  s_particip [as owner] in car {inherited}
    s_aggreg_of
      plate: char[7]
      model: char[8]
  associated_in proj_assignment
    associates project (& employee)
    s_aggreg_of
      hours: float
      delivery: date
  s_aggreg_of
    name: char [15] {inherited}
    phone: char[8] {inherited}
    hire_date: date
    salary: float
    job: char[10]
    id ss_num: char[10] {inherited}
end_class

```

Example 2. Depth level 2 for class 'employee'

Notice that in example 1, the BLOOM schema includes classes for which there is no corresponding relation, they correspond to missing entities detected here as explained later. In example 2, since the interest is on class 'employee' at level 2 in both dimensions, its description in the generalization dimension includes the description inherited from its immediate superclasses, while in the aggregation dimension, the description of its immediate component objects is included even for those inherited ones.

Furthermore, the behavioural description of the class according to the constraints and side effects of the existence dependencies between it and the classes to which it is related, may also be displayed independently of the level of description of its structure. This behaviour results from instantiating the behavioural rules embodied in the metaclasses corresponding to the abstractions involved in the structural description of the class.

In the conversion phase user interaction has also been reduced as much as possible, but it is still needed in two situations. One is in some few cases when there are two possible abstractions for the IND being analyzed and the user must unambiguously which is the appropriate one, and the other one is in the first step of the conversion phase where missing entities are detected and the user must decide in some cases if the entity is relevant.

In the examples given in the rest of this section the BLOOM constructs resulting from the IND being considered in the example, appear in bold, all reserved words of BLOOM are in italics, and comments appear between {}. In the relation schemas, proper identifiers (P_id) are underlined with a double line, while extraneous identifiers (E-id) with a single one. The right hand side of an IND is abbreviated 'rhs' and the left one, 'lhs'.

Step 1: Detection of Missing Entities

In this step all cases of INDs from which missing entities can be detected are analysed thru the following substeps:

Substep 1.1: all INDs on part of key or non key are analyzed here:

- Column 3: Cases $X \subset \text{s.p.CK}$: INDs on simple part of a key
- Column 4: Cases $X \subset \text{c.p.CK}$: INDs on composite part of a key
- Column 5: Cases $X \subset \text{s.NK}$: INDs on simple non key
- Column 6: Cases $X \subset \text{c.NK}$: IND on composite non key

Substep 1.2: all INDs that involve an extraneous identifiers are considered:

- Cases $X \subset E_id$
- Cases $E_id \subset X$

Substep 1.3: every component of a composite key not involved in any IND is analysed.

A *missing entity* is one that conceptually exists but has not been made explicit as a relation in the relational schema. The solution adopted to overcome this anomaly is to create a *virtual relation* for the missing entity which later will be made explicit in the BLOOM schema as a class. The creation of virtual relations has several effects on the augmented relational schema. Next we summarize the set of actions performed in this step:

1.- Create a virtual relation R' with the name of the (set of) attribute(s) in A . In the original relational schema this missing entity is normally referenced by its name, thus, we simply take this name for naming the new relation. These virtual relations will give rise to new classes in next step and in further steps it will receive the same treatment as the other relations.

2.- The P_id of the virtual relation R' is constituted by the (set of) attribute(s) in A . Each attribute in A is renamed for R' by concatenating the first character of its name in R with the word 'name' if the domain is of character type only with letters, 'num' in any other case.

3.- All INDs (simple if A has only one attribute, composite if A has more than one) whose right hand side is A must be transferred to the new virtual relation R' . This action is very important because it *turns INDs corresponding to cases of columns 3, 4, 5 and 6 into cases of columns 1 and 2* as follows:

$X \subset \text{s.p.CK}$ (column 3)	$\rightarrow$	$X \subset \text{SK}$ (column 1) where SK is a P_id
$X \subset \text{c.p.CK}$ (column 4)	$\rightarrow$	$X \subset \text{CK}$ (column 2) where CK is a P_id
$X \subset \text{s.NK}$ (column 5)	$\rightarrow$	$X \subset \text{SK}$ (column 1) where SK is a P_id
$X \subset \text{c.NK}$ (column 6)	$\rightarrow$	$X \subset \text{CK}$ (column 2) where CK is a P_id

and INDs corresponding to cases $X \subset E_id$ into cases $X \subset P_id$.

4.- All INDs (simple or composite) whose left hand side is A must be transferred to the new virtual relation R' . These INDs will turn into INDs whose left side is a simple or composite P_id respectively.

5.- A new IND: $R.A \subset R'.P_id$ is added because now $R.A$ has turn into a foreign key referencing the new virtual relation. Also its inverse $R'.P_id \subset R.A$ is added because R' represents precisely those entities that participate in R , thus, $R.A$ and $R'.P_id$ are the same.

Next some examples of this step are presented.

Example 3: rhs of IND is simple part of a key ($\subset \text{s.p.K}$)

school (<u>university</u> , s_name, head, type);	private_univ.name $\subset$ school.university
private_univ (name, category, owner);	public_univ.name $\subset$ school.university

public_univ (name, category, budget); univ_building.univ $\subset$ school.university
 univ_building (univ_bldg, address, capacity);
 gives rise to: - virtual relation: university (u_name)
 - new IND: school.university $\subset$ university.u_name { $\subset$ P_id}
 university.u_name $\subset$ school.university {inverse: $\subset$ p.K}
 - transfer INDs on school.university to INDs on university-u_name:
 private_univ.name $\subset$ university.u_name { $\subset$ P_id}
 public_univ.name $\subset$ university.u_name { $\subset$ P_id}
 univ_building.univ $\subset$ university.u_name { $\subset$ P_id}

Notice that there were several INDs on school.university (column 3), but as soon as one of them is considered in this step giving rise to the virtual relation 'university', all of them are transferred to the new relation, that is, the original INDs disappear and the new ones which have as rhs university.u_name (column 1) will be treated in a later step which considers the cases of INDs on a simple key.

Example 4: rhs of IND is and extraneous identifier { $\subset$ E_id}

country (name, currency); inflation.currency $\subset$ country.currency
 inflation (currency, year, rate); available.currency $\subset$ country.currency
 available (currency, charge);

gives rise to: - virtual relation: currency (c_name)
 - new INDs: country.currency $\subset$ currency.c_name { $\subset$ P_id}
 currency.c_name $\subset$ country.currency {inverse: $\subset$ E_id}
 - transfer INDs on country.currency to INDs on currency.c_name:
 inflation.currency $\subset$ currency.c_name { $\subset$ P_id}
 available.currency $\subset$ currency.c_name { $\subset$ P_id}

Example 5: no IND for part of a composite key

car_shift (plate_num, emp, day)
 where only 'emp' is involved in an IND: car_shift.emp $\subset$ employee.ss#

Scenario A: experienced user

System: does there exist an object identified only by attribute 'plate_num' independently of 'emp'?

User: yes

Conclusion: attribute 'plate_num' represents a missing entity

Scenario B: naive user

System: does there exist an object identified only by attribute 'plate_num' independently of 'emp'?

User: help (I don't know)

System: car_shift (plate_num, s_name, day)

15467-CA S.Peters monday

15467_CA J.Smith tuesday

does '15467-CA' in these tuples identifies exactly the same object?

User: yes

Conclusion: attribute 'plate_num' represents a missing entity

The conclusion in both cases gives rise to:

- virtual relation: car (c_num)
 - add IND: car_shift.plate_num $\subset$ car.c_num
 car_c_num $\subset$ car_shift.plate_num
 - no transfer of INDs because there are no INDs for this part of the key

All missing entities are detected in the first step of the conversion process in order to avoid the need of performing several passes of the conversion process to treat the INDs (new and transferred) that arise from the creation of virtual relations.

The detection of missing entities is very important not only to make them explicit in the enriched representation, but also for the correct modelling of other classes to which they are connected. If these entities are not detected, then, their relationships with other entities will not be detected either. This leads not only to an incomplete description of a class, but even to an incorrect modelling of the nature of the relationships because a missing IND may lead to a classifying another IND in a case that doesn't correspond to the real semantics, for example, as $o.p.CK \subset X$ instead of $e.p.CK \subset X$.

Step 2 First and Second Columns: rhs of IND is a Key.

In this step all inclusion dependencies on a simple key are treated thru the following substeps:

- Substep 2.1: Case $Key \subset Key$
- Substep 2.2: Case Each Part of Key $\subset Key$
- Substep 2.3: Case Only Part of Key $\subset Key$
- Substep 2.4: Case Non Key $\subset Key$

According to the type of the identifiers, that is, P_id or E_id , of the keys involved in the INDs corresponding to each case, different subcases are distinguished in each substep. Moreover, different semantic situations correspond to each subcase depending on whether the inverse IND exists or not. Examples for some subcases are given next. Only partial descriptions for the classes are given because the purpose is to illustrate only the effect of the IND corresponding to the subcase being analysed.

Example 6: This example illustrates the following subcases of case Simple Key $\subset$ Simple Key:

- $P_id \subset P_id$ without inverse IND,
- $E_id \subset P_id$ without inverse IND,
- $E_id \subset P_id$ with inverse IND and
- $E_id \subset E_id$ without inverse IND.

Relational schema:

Department (d#, dname, manager)
 Project (p#, pname, responsible)
 Project.responsible $\subset$ Department.manager

since it is an IND on an extraneous identifier, in substep 1.2.A where missing entities for E_id 's on the right hand side of INDs are detected, this dependency is analysed and gives rise to:

- virtual relation: Manager (m_name)
- new INDs: Department.manager $\subset$ Manager.m_name { $\subset P_id$ }.....(1)
 Manager.m_name $\subset$ Department.manager {inverse: $\subset E_id$ }...(2)
- transference of INDs on/of Department.manager to Manager.m_name:
 Project.responsible $\subset$ Manager.m_name { $E_id \subset P_id$ }.....(3)

then, in step 1.2.B that detects missing entities for E_id 's on the left hand side of INDs, dependency (3) is considered and the user is inquired about the adequacy of creating a new class for the E_id , that is, for project.responsible. Two possible situations can arise depending on the answer of the user:

- A) If the user answers 'no' then, dependency (3) is treated in substep 2.1 (simple key $\subset$ simple key), subcase $E_id \subset P_id$ where it is converted to:

```
class project
  s aggreg of
    manager [as responsible]
end_class
```

```
class manager
  s particip in
    project [as responsible]
end_class
```

and then IND (1) is treated in substep 2.1, subcase $E_id \subset P_id$ along with its inverse (2) giving rise to:

<pre> class department <u>s:aggreg:of</u> manager end_class [obligatory] </pre>	<pre> class manager <u>s particip in</u> project department end_class </pre>
---	--

B) However, if the user answers 'yes', then there is a missing entity for the E_id and this gives rise to:

- virtual relation: Responsible (r_name)
- new INDs: Project.responsible $\subset$ Responsible.r_name { $\subset P_id$ }.....(4)
 Responsible.r_name $\subset$ Project.responsible { inverse: $\subset E_id$ }...(5)
- transference of INDs on/of Project.responsible to Responsible.r_name:
 Responsible.r_name $\subset$ Manager.m_name { $P_id \subset P_id$ }.....(6)
 { notice that the original IND has been transferred first to (3) and then to (6) }

then, IND (6) is treated in substep 2.1 (simple key $\subset$ simple key), subcase $P_id \subset P_id$ resulting in:

<pre> class manager <u>generaliz of</u> responsible <u>s aggr of</u> m_name end_class </pre>	<pre> class responsible <u>specializ of</u> manager end_class </pre>
--	--

and finally IND (4) is analyzed in substep 2.1, subcase $E_id \subset P_id$ along with its inverse (5), as well as, IND (1) and its inverse (2). The result of converting INDs (1) and (2) is the same as in A) while (4) and (5) are converted to:

<pre> class project <u>s aggr of</u> responsible end_class </pre>	<pre> class responsible <u>specializ of</u> manager <u>s particip in</u> project [obligatory] end_class </pre>
---	--

Example 7: this example illustrates one of the possible semantic situations for substep 2.4 where INDs of type non key $\subset$ simple key are analysed.

employee (ss#, name, salary)
 vehicle (car#, model, owner) where owner is non key and not null
 vehicle.owner $\subset$ employee.ss#

this IND is converted to:

<pre> class employee <u>s partic as set in</u> vehicle end_class </pre>	<pre> class vehicle <u>s aggreg of set of</u> employee [as owner] end_class </pre>
---	--

Example 8: another semantic situation for substep 2.4 where case non key $\subset$ simple key is considered.

club (c_name, d_founded, site)
 person (p_id, name, club); (in this example a person can be member of only one club)
 club.c_name $\subset$ person.club

the corresponding abstraction for this IND is the collection aggregation:

```
class club
  collection_of person
end_class
```

```
class person
  collected_in club
end_class
```

Space limitations prevent us from illustrating other subcases, but the ones presented here can give a feeling of the conversion phase. More examples as well as the algorithms are discussed in [Cas92].

6. Conclusions.

We have presented here a methodology for upgrading the semantic level of relational database schemas. It comprises not only the conversion of these schemas to richer ones expressed in an adequate canonical model, but also the knowledge acquisition process that analyses both, the schema and the extension of the database, to extract implicit semantics. Emphasis has been put on the efficiency aspect because analysing extensions is a costly operations, and also in minimizing the user intervention so that the methodology can be the basis of a tool that automatizes the whole enrichment process as much as possible. The algorithms to implement the ideas exposed in this paper are implemented in C on a SUN SS-10 workstation.

Since the enrichment process is situated in the context of interoperability, it not only elicits the semantics of the databases, but at the same time it prepares the schemas for being useful in the integration of schemas, in particular, for facilitating the detection of interdatabase semantic relationships [GarCasSal92]. It is part of a methodology for schema integration where semantic enrichment is essential and the semantic abstractions resulting from the enrichment play a central role.

Now we are working on the other phases of the methodology, that is, the detection and merging phases, and plan to build a tool that semiautomatizes the whole integration process. Also we plan to extend the enrichment with additional information extracted from other sources, mainly, of operational kind.

Acknowledgements.

I am undoubtedly grateful to Fèlix Saltor for the so useful discussions held with him all thru the development of the work reported here, as well as for his comments on earlier versions of this paper. I am also grateful to Manuel García for his suggestions and for his help in the edition of this paper and for. This work has been partially supported by the Spanish PRONTIC programme, under project TIC89/0303.

References.

- [BatLenNav86] Batini, Lenzerini & Navathe: "A Comparative Analysis of Methodologies for Database Schema Integration". ACM Computing Surveys, Vol.18, No.4, Dec.1986.
- [Cas92] M.Castellanos: "Algorithms for Semantic Enrichment". Dept. LSI, UPC. In preparation.
- [CasAma83] M.Casanova, J.Amarel: "Designing ER Schemas from Conventional Information Systems". Proceedings, third International ER Conf. 1983.
- [CasSal91] M.G.Castellanos, F.Saltor: "Semantic Enrichment of Database Schemas: an Object Oriented Approach". In [Kam91].
- [CasSalGar91] M.G.Castellanos, F.Saltor & M.García-Solaco: "The Development of Semantic Concepts in the BLOOM Model using an Object Metamodel". Report LSI-91-22. U.P.C., Barcelona.
- [Cod79] E.F.Codd: "Extending the Database Relational Model to Capture More Meaning". ACM TODS, vol 4, #4 (Dec 1979).
- [DavAro87] Davis & Arora: "Converting a Relational Database Model into an ER Model". In [Mar88].

- [GarCasSal92] M.García, M.Castellanos & F.Saltor: "Discovering Interdatabase Resemblance of Classes for Interoperable Databases". To appear in: Proceedings 2nd Int. Workshop on Interoperability in Multidatabase Systems (RIDE IMS-93, Vienna). IEEE-CS Press.
- [GarSal91] M.García & F.Saltor: "Discriminated Operations in Interoperable Databases". In [Kam 91].
- [HamMcL78] Hammer & McLeod: "The Semantic Data Model: a modelling mechanism for database applications". Proc. of ACM SIGMOD Conf. on Management of Data. 1978.
- [JohKal98] Johannesson & Kalman: "A Method for Translating Relational Schemas into Conceptual Schemas". Proc. 8th Int. ER Conference, Toronto. 1989.
- [Kam91] Y.Kambayashi, M.Rusinkiewicz & A.Sheth (eds.): Proceedings, First International Workshop on Interoperability in Multidatabase Systems (Kyoto). IEEE-CS Press, 1991.
- [ManRai88] Heikki Mannila & K.Raiha: "Algorithms for Inferring Functional Dependencies". Report A-1988-3. Department of Computer Science, University of Tampere, april 1988.
- [Mar88] S.March (ed.): Proc. 6th Int.Conf. on Entity-Relationship Approach. N.Y. 1987.
- [MarMak90] Markowitz & Makowsky: "Identifying Extended ER object Structures in Relational Schemas". IEEE Trans. on Software Engineering. Vol.16, No.8 (august 1990).
- [NavAwo87] Navathe & Awong: "Abstracting Relational and Hierarchical Data with a Semantic Data Model". In [Mar88].
- [SalCasGar91] F.Saltor, M.G.Castellanos & M.García-Solaco: "Suitability of Data Models as Canonical Models for Federated Databases". ACM SIGMOD Record. Vol.20, No.4 (Dec. 1991)
- [SalCasGar92] F.Saltor, M.Castellanos & M.García: "Overcoming Schematic Discrepancies in Interoperable Databases". In Hsiao, Neuhold & Sacks-Davis (eds.) Semantics of Interoperable Database Systems (Proceedings, DS-5 Conference, Lorne). Elsevier North-Holland, 1992.
- [She91] A.Sheth (ed.): Special issue on Semantic Heterogeneity, ACM SIGMOD Record. Vol.20, No.4 (Dec. 1991).
- [SheLar90] A.Sheth & J.Larson: "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases". ACM Computing Surveys. Vol.22, No.3 (Sept. 90).

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

List of research reports (1993).

LSI-93-1-R "A methodology for semantically enriching interoperable databases", Malú Castellanos.

LSI-93-2-R "Extraction of data dependencies", Malú Castellanos and Fèlix Saltor.

LSI-93-3-R "The use of visibility coherence for radiosity computation", X. Pueyo.

LSI-93-4-R "An integral geometry based method for fast form-factor computation", Mateu Sbert.

LSI-93-5-R "Temporal coherence in progressive radiosity", D. Tost and X. Pueyo.

LSI-93-6-R "Multilevel use of coherence for complex radiosity environments", Josep Vilaplana and Xavier Pueyo.

Internal reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics (U.P.C.)
Pau Gargallo 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es