

Overcoming Schematic Discrepancies in Interoperable Databases*

F. Saltor, M. G. Castellanos and M. García-Solaco

Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Pau Gargallo 5, E-08028 Barcelona. {saltor,castellanos,mgarcia}@lsi.upc.es

Abstract

An important kind of semantic conflicts arising in database interoperability are schematic discrepancies, where data in one database correspond to metadata in another. We present a general solution to these discrepancies, based on a framework of two dimensions: generalization and aggregation. Operations to transform metadata into data and vice versa are defined, in the relational model and in an object-oriented model. These operations may be applied at different levels in a federated architecture.

Keyword Codes: H.2.5

Keywords: Heterogeneous Databases

1. INTRODUCTION

The cooperation between autonomous and already existing databases in order to share their data while at the same time maintaining their autonomy has given rise to a new kind of distributed database architecture called Interoperable or Federated Database Systems [1] (FDBS). Heterogeneity is one of their features and is an effect of their design and organization autonomies. On one side, we have a syntactic heterogeneity that stems from the fact that each database may be implemented in a different DBMS with a different data model. On the other, even if they use the same data model, a real world situation may have been represented in different ways by the different designers, giving as consequence semantic conflicts. These heterogeneities have to be solved when different databases are associated into a FDBS.

Semantic heterogeneities are difficult to overcome, they must be detected and solved thru an integration process where a federated schema is derived from the component ones. In this paper our concern is on one kind of semantic conflicts called *schematic discrepancies* where data in one database correspond to metadata in another.

Here we present a general solution to solve schematic discrepancies by applying special operations to transform data into metadata and vice versa. We show its usage in the interoperability context by applying the operators for the conformation of schemas in the two upper levels of the federated architecture in [1]. On one side, for the integration of export schemas into a federated one and on the other for the derivation of external schemas from a federated one.

* This research has been partially supported by the spanish PRONTIC programme under project TIC89/0303.

This paper is organized as follows. Section 2 exposes the problem thru an example that will be used throughout the rest of the paper. Section 3 develops the operators in the relational model, and section 4 presents their object-oriented counterparts. In section 5 we discuss some aspects of our approach, and finally in section 6 we present our conclusions.

2. WHAT'S THE PROBLEM? SCHEMATIC DISCREPANCIES

Let us consider an example that we borrow from the one used in [2]. It is related to the stock market and considers three stock databases: one from New York, another from Barcelona and another from Melbourne. In all of them there is information about the closing price of each stock per day. The schemas of the databases are the following ones:

Database *NY*

(one relation with one tuple per day per stock with its closing price)

<i>S:</i>	<u><i>date</i></u>	<u><i>stock</i></u>	<i>clsprice</i>
	910408	IBM	347
	910408	HP	418
	910408	GM	250
	910409	IBM	350
	910409	HP	420
	910409	GM	215

Database *Barcelona*

(one relation with one tuple per day and one attribute per stock whose value is the closing price of that stock)

<i>S:</i>	<u><i>date</i></u>	<i>HP</i>	<i>IBM</i>	<i>GM</i>
	910408	418	365	250
	910409	420	350	200

Database *Melbourne*

(one relation per stock with one tuple per day)

<i>HP:</i>	<u><i>date</i></u>	<i>clsprice</i>	<i>IBM:</i>	<u><i>date</i></u>	<i>clsprice</i>	<i>GM:</i>	<u><i>date</i></u>	<i>clsprice</i>
	910408	425		910408	347		910408	385
	910409	420		910409	350		910409	320

In the previous example of multiple databases it can be appreciated that the values of the attribute 'stock' in database NY corresponds to:

- names of attributes in database Barcelona
- names of relations in database Melbourne

date in database NY corresponds to *metadata* in databases Barcelona and Melbourne (value discrepancies are not considered in this paper).

This kind of differences where data in one database correspond to metadata in another are called *Schematic Discrepancies*. They appear frequently in the interoperability context due to

the independence of the design process and the preservation of database autonomy. In our example, the databases have the same purpose, that is, to provide information related to the closing prices of the different stocks they deal with (notice that each database may deal with a different set of stocks). However, if databases NY, Barcelona and Melbourne are to be associated into a FDBS their schemas have to be *integrated* into a federated one. This integration is difficult to achieve because schematic discrepancies have to be solved in order to obtain a single unified structure. One possible solution is to unify all the schemas to a single form and then integrate them. For unifying their structures, some transformations have to be applied to the schemas so that metadata can turn into data and vice versa.

Once the federated schema is obtained, the problem is not yet completely solved since its structure may not be convenient for all users. It can be the case that a user wants to work at the FDBS level with a schema with the form of the one he was used to work with before entering the federation. In particular, he wants to continue viewing as data what he is used to view as data, and as metadata what was viewed as metadata. To satisfy this requirement, one possible solution could be to derive for him an *external schema* by derivation from a federated schema.

From the above discussion, we can conclude that a mechanism to transform structures by converting data into metadata and metadata into data could be very useful. Our approach follows this line and provides a set of operations for performing these transformations. They can be applied either for integration or for derivation of external schemas. In the two following sections we present the set of operations in a relational context and in an object-oriented one.

3. SCHEMA TRANSFORMATIONS IN THE RELATIONAL MODEL

As discussed in the previous section, we need to have operations to convert metadata into data values and vice versa. Since the basic Relational Model does not provide such operations, we extend the model with additional operations.

We first define operations along the *Generalization/Specialization* dimension, that will solve schematic discrepancies such as those of Melbourne with NY in our example, and then operations along the *Aggregation/Decomposition* dimension, that will solve discrepancies such as those of Barcelona with NY.

We adopt for our exposition a framework based on these dimensions because it is a very general, model independent framework that facilitates the correspondence of the extended relational operations with those of an O-O model (section 4), and will help in showing the complementarity among dimensions (see section 5).

3.1 Operations along the Generalization dimension

We can observe that the schema of the Melbourne database is a *specialization* of the schema of the NY database, in the sense that each relation in Melbourne corresponds to a subclass (partition) of the relation in NY. conversely, the relation in NY can be seen as a *generalization* (superclass) of the relations in Melbourne. To solve this kind of schematic discrepancies, we need operations along the generalization/specialization dimension.

3.1.1 Discriminated union

a) Definition

Given two or more relations with the same schema, or with union compatible schemas, the (inner) union operation of the relational model produces a new relation with the same schema, and with an extension that is the set union of the extensions of the operands.

Our (inner) *discriminated union* operation is also constrained to apply only to union compatible relations; its result is a new relation with an extended schema, which has an additional attribute. This attribute has the name *discriminant*, and shows, for each tuple in the result, from which of the operands it comes; by default, it takes as values the names of the relations. The extension of the result is formed by first multiplying the extension of each operand by its name, and then unioning these products. In case an operand has no name, because it is the result of a subexpression, the system assigns a unique name to it.

Note that, in contrast with the union operation, our discriminated union involves no removal of duplicates, because no duplicates can be produced, due to the discriminant.

The discriminated union operation was introduced in [3]. As already noted there, if there is an operation that given a relation R, returns a relation with just one attribute and one tuple: the name of R as a string, such as the 'NOTE' operation of RM/T [4], then the discriminated union, noted \cup , can be defined as:

$$R \cup S = ((\text{NOTE}(R) \times R)) \cup ((\text{NOTE}(S) \times S))$$

In case an operand is an expression, the value of the discriminant will be a system generated string.

This operation transforms *metadata*: the names of the operand relations, into *data*: the values of the discriminant.

b) Uses for database interoperability

In the context of interoperability, the discriminated union can be applied at three distinct levels of the architecture:

b.1) Intra Database Discriminated Union

Going back to our example, if a federated schema FNY is to have a relation schema like the one of S in NY, then the schema of the Melbourne database is transformed into the appropriate schema precisely by the discriminated union (*intra database discriminated union*) of its three relations HP, IBM and GM. The discriminant attribute may be renamed, if so desired, as in this example where its name changes from *discriminant* to *stock*. The resulting relation is shown next:

S:	<u>date</u>	<u>stock</u>	<u>clsprice</u>
	910408	HP	425
	910409	HP	420
	910408	IBM	347
	910409	IBM	350
	910408	GM	385
	910409	GM	320

b.2) Inter Database Discriminated Union

Once the Melbourne schema is conformed, it can be easily integrated with the NY schema to produce the desired federated schema FNY. One way to perform this is to apply a variant of the discriminated union operation to operands which are tables of different databases; in this case the values of the discriminant attribute will be the names of the component databases (*inter database discriminated union*). This has a number of advantages, explained in [3], including the support of multiple semantics in one federated schema. Users not needing to know the sources of the data would be provided with an external schema that projects out this DB discriminant (database transparency).

b.3) Intra Federation Discriminated Union

Let us assume that a federated schema FM is to have the form of the Melbourne relations, and that the NY schema is transformed into this form (by the partition by attribute operation, see below) and integrated. There may be a need to provide previous users of NY with an external schema with the form of S in NY. This is accomplished by applying the discriminated union operation to the federated schema FM, and by renaming the discriminant attribute (*intra federation discriminated union*).

c) Other usage

The discriminated union operation may be used in stand alone databases, too: in applications, in view definition, and by the optimizer, as explained in [3].

d) Outer discriminated union

The (inner) discriminated union operation cannot be applied to relations that are not union compatible. The *outer discriminated union* may always be applied, and generates null values similarly to the outer union of the relational model.

This operation would be applied as needed (intra database or inter database), in case of relations that are not union compatible.

3.1.2 Partition by Attribute

a) Definition

Given a relation R and one of its attributes, A, let a be a value of A present in the extension of R. We may restrict (select) R with the predicate $A=a$, project out the A attribute, and give the name a to the resulting relation. The *partition by attribute* operation does precisely this for each value a of A present in the extension of R. The result is a set of named relations, all having the same schema, namely the schema of R without A. A is called the *partitioning attribute*.

Note that this operation is not algebraic, because its result is not one relation, and, even in the case of all tuples of R having the same value of A, because it names the resulting relation.

This operation is the inverse of the discriminated union: applying it to the result of the discriminated union of a set of relations, with the discriminant as partitioning attribute, produces the original relations. Conversely, given a relation, if we partition it and then apply the discriminated union to its result, we obtain the original relation.

This operation is a generalization of the "split" operation of [5], and is similar to the "partition by attribute" operation of [4]; the difference is in giving names to the resulting relations. It has some resemblance to the framing operation called "partitioning a relation by individual values" in [6], but it produces a set instead of a relation, and it assigns the existing value a as a name, and not a system generated value as a frame identifier.

This operation transforms *data*: the values of the partitioning attribute, into *metadata*: the names of the resulting relations.

b) Uses for database interoperability

This operation may be applied at two levels of the federated architecture.

b.1) Intra Database Partition by Attribute

In the case of the federated schema FM that is to have a relation schema with the form of the Melbourne schema, the schema of the NY database is transformed into the appropriate schema precisely by the partition by attribute operation, with *stock* as partitioning attribute (*intra database partition by attribute*).

Once the NY schema is so conformed, it can be integrated with the Melbourne schema to produce the desired federated schema FM; this may be done by applying the discriminated union operation to each pair of corresponding relations.

b.2) Intra Federation Partition by Attribute

In the case of the federated schema FNY that is to have the form of the NY relation, the Melbourne schema is transformed into this form by the discriminated union operation, as seen before, and integrated. There may be a need to provide previous users of Melbourne with an external schema with the form they are used to. This is accomplished by applying the partition by attribute operation to the federated schema FNY, with *Stock* as partitioning attribute (*intra federation partition by attribute*).

c) Other usage

The partition by attribute operation corresponds to one case of (primary) horizontal fragmentation in distributed databases [7], and may be used for this purpose. The names of the resulting relations are not as relevant in this case.

3.2 Operations along the Aggregation dimension

A schema such as the one of the Barcelona database of the example is an *aggregation* of NY, in the sense that each tuple in Barcelona is the (cartesian) aggregation of values in NY (of tuples in NY with the same date). Reciprocally, tuples in NY can be seen as *decompositions* of those in Barcelona.

To solve this kind of schematic discrepancies, we need operations along the aggregation/decomposition dimension.

3.2.1 Decomposition

a) Definition

Given a relation R with (simple) key K and non-key attributes A_1, A_1, \dots, A_n , each tuple in R may be decomposed into n triplets formed by the value of K, the name A_i , and the value of A_i , for $i=1$ to n. Each such triplet has the schema *R Key - Attribute - Value*. The *decomposition* operation, applied to R, transforms each tuple of R into its n triplets, and produces a relation Dec(R) with schema (R Key, Attribute, Value), and with an extension formed by all those triplets.

The domain of R Key is the same as that of K. The domain of Attribute is the set of strings valid as attribute names. The domain of Value is the union of the domains of attributes A_1 to A_n . The key of Dec(R) is composed of R Key (the key of R) and Attribute.

The cardinality of Dec(R) is n times the cardinality of R. In case some value in R was null, the corresponding triplet in Dec(R) will have null in the Value attribute; if these triplets are not desired, they can be eliminated by restriction.

The triplets R Key - Attribute - Value correspond to those of the Entity Set Model of [8], which may be considered a predecessor of the relational model. Similarities exist with the

representation of the Taxis model. The First Order Normal Form of [9] resembles our Dec(R), although they decompose the key, too, corresponding to our Full decomposition (see below).

This operation transforms *metadata*: the names of the non key attributes of R, into *data*: the values of the *Attribute* attribute of Dec(R).

b) Uses for database interoperability

This operation may be applied at two levels of the federated architecture.

b.1) Intra Database Decomposition

Referring to our example, if a federated schema FNY is to have a relation schema like the one of S in NY, then the schema in the Barcelona database is transformed into the appropriate schema precisely by the decomposition operation (*intra database* decomposition). Once so transformed, the schema may be integrated with the NY schema, for example by the discriminated union operation.

b.2) Intra Federation Decomposition

In case what is wanted is a federated schema FB with the form of the Barcelona relation, and the NY database schema is transformed into this form (by the composition operation, see below) and integrated, previous users of NY will need an external schema with their usual form. The decomposition operation will perform this work (*intra federation* decomposition).

c) Other usage

People in favour of binary models, who dislike n-ary relations and insist in only binary relations, may use the decomposition operation and then the partition by attribute operation - with *Attribute* as the partitioning attribute- to produce their desired schema.

The decomposition operation may also be used to decompose relations into a form corresponding to a decomposed implementation. In this case, a full decomposition (see below) may be preferable.

d) Full decomposition

The decomposition operation, as defined in a), decomposes only non key (non prime) attributes. In some cases it may be interesting to decompose all attributes, particularly if the key of R is composed, in which case the decomposition operation was not defined.

This is performed by the *full decomposition* (FDec) operation. It decomposes each tuple of a relation R into d tuples, where d is the degree of R. It also generates a unique tuple identifier (tid) for each tuple in R, and places this tid in the generated tuples. Each one of the d tuples in FDec(R) generated from a given tuple T in R is composed of the tid assigned to T, the value of the key of T (simple or composed), the name of one attribute (prime or non prime), and the value of this attribute in T. The schema of FDec(R) is then (*Tid-R Key attribute(s)-Attribute-Value*). Its degree is then three plus the number of attributes in the key of R.

There are two alternate keys in FDec(R): *Tid - Attribute*, and *R Key attributes -Attribute*, since each tuple in R is identified both by its tid and by its key. In case one of these two identifiers is not desired, it can be filtered out by projection.

If the key of R is composed, there will be several *R Key attributes* in R. In a context of a relational model extended with nested or NF2 relations [10], these attributes can be combined into one complex attribute by a *nest* operation (called "group" in [5]), so that *R Key* has always just one attribute.

3.2.2 Composition

a) Definition

Assume a relation R with a composite key K and only one non key attribute A , and let K_1, K_2, \dots, K_k be the components of K . Let us designate K_i as the *composing* attribute. For a given value of $K-K_i$ appearing in R , we may group all tuples that have this value of $K-K_i$ into one group, and aggregate them into a tuple, formed by this common value of $K-K_i$, and the values of A of all tuples in the group. The schema of this resulting tuple will consist of all K_j except the composing attribute K_i , plus as many attributes as members of the group, that will receive as names the *values* of the K_j attribute in their respective tuples in R .

Performing this composition for each value of $K-K_i$ present in R , produces the *composition* by K_i of R , noted $\text{Comp}\langle K_i \rangle(R)$.

Each of the tuples in $\text{Comp}\langle K_i \rangle(R)$ may have a different schema, since the values of K_j for each of the groups may be different. Therefore, the schema of the resulting relation will be the set union of the schemas of its tuples, i.e. $K-K_i$ plus all values of K_j appearing in R . Each tuple in $\text{Comp}\langle K_i \rangle(R)$ will have a null value for an attribute that was not present in its group as value of K_j .

The key of the resulting relation $\text{Comp}\langle K_i \rangle(R)$ is $K-K_i$. Its cardinality will be less than that of R , or equal if $K-K_i$ functionally determines K_i (and then R is not in 2nd Normal Form). The domains of non key attributes will be the domain of A in R .

We know of no similar operation in the literature. This operation is the inverse of decomposition; the inverse of composition is decomposition followed by elimination of those tuples that have null values generated by composition.

This operation transforms *data*: values of K_j , into *metadata*: names of the non key attributes of $\text{Comp}\langle K_i \rangle(R)$.

b) Uses for database interoperability

This operation may be applied at two levels of the federated architecture.

b.1) Intra Database Composition

Referring to our example, if a federated schema FB is to have a relation schema like the one of Barcelona, then the schema in the NY database is transformed into the appropriate schema precisely by the composition operation (*intra database* composition): $\text{Comp}\langle \text{Stock} \rangle(S)$. Once so transformed, the schema may be integrated with the Barcelona schema, for example by the discriminated union operation.

b.2) Intra Federation Composition

In the case of a federated schema FNY with the form of the schema of the NY database, the Barcelona schema is transformed by the decomposition operation and integrated, as seen above. An external schema for Barcelona users, with the form of the Barcelona schema, may be produced by the composition operation: $\text{Comp}\langle \text{StkCode} \rangle(FNY)$.

c) Other usage

This operation may be used to recompose tuples or relations implemented as decomposed.

d) Nested composition

If our relational model is extended to include nested relations (NF2), then we can define a composition operation that produces a relation with just two attributes: the Key and the Value. The *Key* will be a nested attribute, composed of $K-K_i$ (flat, i.e. unnested, only if $k=2$). The *Value* attribute will be a nested attribute, too, composed of all values of the composing attribute.

4. SCHEMA TRANSFORMATIONS IN AN OBJECT-ORIENTED MODEL

As stated in [11], the characteristics of object-oriented models make them ideal candidates as canonical models for interoperable databases. Henceforth, we have developed the corresponding object-oriented counterpart for those extended operations presented in the previous section. In particular, we have added these O-O operations to the BLOOM model [12] that we have designed bearing in mind its suitability as canonical model, but they can be added to any other object model. In this section we make use of BLOOM only for illustrating the examples. First, we present the BLOOM representation corresponding to the relational schemas of the example in section 2 (it would be the result of a semantic enrichment [13] applied to the relational schemas, even though the example is so simple that the enrichment is reduced to a straightforward conversion). Then, we explain the operations on the generalization/specialization and aggregation dimensions.

Object Base *NY*:

```
class S
  simple_aggreg_of
    date: date
    stock: string
    clsprice: float HP: float
end_class
```

Object Base *Barcelona*:

```
class S
  simple_aggreg_of
    date: date
    IBM: float
    GM: float
    ...
end_class
```

Object Base *Melbourne*:

```
class IBM          class HP          class GM
  simple_aggreg_of  simple_aggreg_of  simple_aggreg_of
    date: date      date: date          date: date
    clsprice: float clsprice: float      clsprice: float
end_class          end_class          end_class
```

4.1. Operations in the generalization/specialization dimension

In a way analogous to the generalization/specialization dimension in the relational context, we define two inverse operations for the object-oriented context.

4.1.1. Discriminated generalization (counterpart of discriminated union)

a) Definition

Given two or more classes, the generalization operation produces a new one, the superclass, from the common part of its subclasses. Our discriminated generalization is an extension of the normal generalization where the structure of the superclass is extended with a discriminant attribute that takes as values the names of the subclasses. The discriminant must be given a name, in fact, it is the only attribute that must be specified when the superclass is being

defined. At the instance level, each object from a subclass is also an object of the superclass where it incorporates the name of the subclass to which it belongs.

b) Usage for interoperability

To transform the representation of object base Melbourne where there is a class for each stock, to the representation of object base NY where there is only one class with an object (instance) for each stock, we define the following class.

```
class S
  discr_gen_of IBM, HP, BA [stock]      ('stock' is the name of the discriminant attribute)
end_class
```

Class S is the discriminated generalization of the classes IBM, HP and BA from object base Melbourne. Its structure is upward inherited from the common structure of her subclasses plus the discriminant attribute 'stock' whose domain is constituted by the names of its subclasses:

```
class S
  discr_gen_of IBM, HP, GM
  simple_aggr_of
    stock: {'IBM', 'HP', 'GM'}
    date: date
    clsprice: float
end_class
```

Analogously to the relational context, here we also have three distinct levels of application of the discriminated generalization, namely intra-database, inter-database and intra-federation

c) Outer discriminated generalization

Sometimes it is convenient that the structure of the superclass be conformed not only by what is common to its subclasses, but also by their specific parts. The objects in the superclass to which a property (from another sibling subclass) does not apply will have a null for it.

For our example, let's suppose that class IBM in object base Melbourne has an specific attribute 'motif' and class GM has an specific attribute 'items'. A class with a similar structure as the one of class S in NY but that includes all attributes of the subclasses is defined by applying the outer discriminated generalization:

```
class S
  outer_discr_gen_of IBM,HP,GM [stock]
end_class
```

the resulting structure of this new class is the following one:

```
class S
  discr_gen_of IBM, HP, BA
  simple_aggreg_of
    common
    stock: {'IBM', 'HP', 'GM'}
```

```

        date: date
        clsprice: float
    specific
        motif [IBM]: string
        items [BA]: integer
end_class

```

This structure of S permits to obtain a view of all the attributes involved in the subclasses.

4.1.2. Specialization by Attribute

a) Definition (counterpart of partition by attribute)

The specified class is specialized into as many subclasses as there are different values for the attribute specified as the *specializing* attribute, thus, it is an *alternative* specialization (in the sense of BLOOM) where all objects that have the same value for this attribute constitute the instances of the new subclass whose name is precisely this value. This operation is the inverse of the discriminated generalization: applying it to the result of a discriminated generalization of a set of classes, using the discriminant as specializing attribute, the original (sub)classes can be obtained.

b) Usage for interoperability

To transform the representation of object base NY where there is only one class S with an object for each stock, to the representation of object base Melbourne where there is one class for each stock, we define class S as follows:

```

class S
    specialized_by_attr [stock]
end_class

```

The resulting classes would have the following structures:

<pre> class IBM specializ_of S simple_aggreg_of date: date clsprice: float end_class </pre>	<pre> class HP specializ_of S simple_aggreg_of date: date clsprice: float end_class </pre>	<pre> class BA specializ_of S simple_aggreg_of date: date clsprice: float end_class </pre>
---	--	--

Each class is obtained by specializing class S by the attribute 'stock'. The subclasses inherit the structure of the superclass, except the discriminant attribute 'stock'.

4.2. Operations in the Aggregation dimension

We present here the counterparts of those relational operations in the aggregation dimension explained in subsection 3.2.

4.2.1. Decomposition

a) *Definition* (counterpart of the operation with the same name in relational)

Each object p of the specified class P is decomposed into several objects of a new class N so that each non identifier attribute of p gives rise to a new object of N . The structure of N is an aggregation (*simple aggregation* in the sense of BLOOM) of three attributes: one attribute (A1) is the identifier of class P and two new attributes: one of them (A2) takes its values from the names of the non identifier attributes of P , and the other one (A3) has as domain the union of the domains of the non identifier attributes of P . These attributes must be given a name. The identifier of the new class N is a composite identifier composed of the identifier of P and attribute A2.

b) *Usage for interoperability*

To transform the representation of object base Barcelona where there is one class with an attribute for each stock, to the representation of object base NY where there is only one class with an object for each stock, we define the following class:

```
class R
  decomposition_of S [stock, clsprice]      ('stock' is the name for the new attribute A2 and
end_class                                  'clsprice' is the name for the new attribute A3).
```

Class R results from applying the decomposition operation to S , and its structure is exactly the one of class S in object base NY.

4.2.2. Composition

a) *Definition* (counterpart of the operation with the same name in relational)

For classes with a composite identifier ID where I_i is one of its components and one non identifier attribute A , it is possible to group all objects with the same value for $(ID - I_i)$ into just one new object composed by the aggregation of the common value of $(ID - I_i)$ and the values of attribute A of all objects grouped. Thus, the structure of the new class is given by the aggregation of all attributes of ID except I_i , plus as many attributes as there are different values for I_i named according to these values. When the new class is being defined, we specify I_i in brackets.

b) *Usage for interoperability*

To transform the representation of object base NY where there is only one class with an object for each stock, to the representation of Barcelona where there is one class with an attribute for each stock, the following class is defined:

```
class R
  decomposition_of S [stock]                ('stock' is the attribute of S whose values will
end_class                                  turn into names of new attributes of R)
```

The resulting structure of the new class is just the same one of class S in object base Barcelona.

5. DISCUSSION

Once we have seen our proposed operations, we can discuss about their usage to solve schematic discrepancies in interoperable databases.

5.1. Complementariness among dimensions

Schematic discrepancies may appear along the generalization/specialization dimension, and we have shown how to solve them using the discriminated union and partition by attribute operations, in the relational context, and discriminated generalization and specialization by attribute, in object orientation.

Another kind of schematic discrepancies may exist along the aggregation/decomposition dimension, and this paper has shown their solution by the extended relational decomposition and composition operations, and by object-oriented decomposition and composition.

Schematic discrepancies may appear along the *classification/instantiation* dimension, too. This is not apparent in our example, and we do not cover them in this paper.

We claim that no other kind of schematic discrepancies (in this sense of data - metadata conflict) may exist, because these three dimensions are complementary and cover all data modelling space. For instance, the schema of database Barcelona may be transformed into the form of Melbourne by decomposition followed by partition by attribute.

5.2. Comparison with other approaches

Schematic discrepancies are not dealt with in the literature. The only approach we know of is [2], that is more oriented towards a loosely coupled federation, whereas ours is oriented towards a tightly coupled one. We feel that our solution is easier, because our operations are simpler than their "higher order expressions", as effective, because we can solve the same problems they show. In some sense it is more general because we do not restrict ourselves to a relational or extended relational context, as we have shown in section 4. However, we have not covered yet the problem of updates considered by them.

6. CONCLUSIONS

We have shown our approach to solve one kind of semantic discrepancies in interoperable databases, namely *schematic discrepancies*, where data in a database correspond to metadata in another.

By defining extended relational operations: *discriminated union* and *partition by attribute*, along the generalization/specialization dimension, as well as *decomposition* and *composition*, along the aggregation/decomposition dimension, metadata may be transformed into data, and vice versa, in the relational context. These operations may be applied at different levels of a federated architecture, to solve different cases of schematic discrepancies.

Similarly, in an object-oriented model, *discriminated generalization* and *specialization by attribute*, along the first dimension, as well as *decomposition* and *composition*, along the second, solve schematic discrepancies.

Our solution appears simpler than the only other known approach, that of [2], and it solves at least the same problems of schematic discrepancies, as we have shown by using their example. In some sense it is more general, because it is placed in a framework based on two dimensions, and not restricted to a relational model.

We still have to cover the problem of updates on the views of the federated schemas conformed by applying our operations, as well as to refine the effects of the operations in the object oriented version.

References

- 1 A.Sheth and J.Larson: "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases". *ACM Computing Surveys*, Vol.22, No.3 (1990).
- 2 R.Krishnamurthy, W.Litwin and W.Kent: "Language Features for Interoperability of Databases with Schematic Discrepancies". *Proceedings, First International Workshop on Interoperability in Multidatabase Systems*. Kyoto. IEEE-CS Press (1991).
- 3 M.García-Solaco and F.Saltor: "Discriminated Operations for Interoperable Data-bases". *Proceedings, First International Workshop on Interoperability in Multidatabase Systems*. Kyoto. IEEE-CS Press (1991).
- 4 E.F.Codd: "Extending the Database Relational Model to Capture More Meaning" *ACM TODS vol4*, No.4 (1979).
- 5 G.Gardarin and P.Valduriez: *Relational Databases and Knowledge Bases*. Addison-Wesley (1989).
- 6 E.F.Codd: *The Relational Model for Database Management, Version2*. Addison-Wesley (1990).
- 7 M.T.Ozsu and P.Valduriez: *Principles of Distributed Database Systems*. Prentice Hall (1991).
- 8 Senko, Altman, Astrahan and Fehder: "Data Structuring and Access in Database Systems: II Information Organization". *IBM Systems Journal*, vol 1, pages 45-63 (1973).
- 9 W.Litwin, M.Ketabchi and R.Krishnamurthy: "First Order Normal Form for Relational Databases and Multidatabases". *ACM SIGMOD Record*, Vol.20, No.4 (1991).
- 10 H-J.Schek and M.Scholl: "The Relational Model with Relation-Valued Attributes". *Information Systems*, Vol 11, No.2 (1986).
- 11 F.Saltor, M.G.Castellanos and M.García-Solaco: "Suitability on Data Models as Canonical Models for Federated Databases". *ACM SIGMOD Record*, Vol.20, No.4 (1991).
- 12 M.G.Castellanos, F.Saltor and M.García-Solaco: "A Canonical Model for the Interoperability Among Object-Oriented and Relational Databases". *Proceedings, International Workshop on Distributed Object Management*. Edmonton (1992).
- 13 M.G.Castellanos and F.Saltor: "Semantic Enrichment of Database Schemas: an Object-Oriented Approach". *Proceedings, First International Workshop on Interoperability in Multidatabase Systems*. Kyoto. IEEE-CS Press (1991).