Discriminated Operations for Interoperable Databases

Manuel García-Solaco Fèlix Saltor

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona
eamgarci@ebrupc51.bitnet easaltor@ebrupc51.bitnet

Abstract

Federation Rules to be followed when associating external schemas into federated schemas are formulated. Discriminated operations in an object oriented model, such as discriminated generalization, and in the relational model, such as discriminated union and outer union, are introduced. It is then shown how they may help in obeying the Federation Rules. These operations allow multiple semantics in a single federated schema, and thence minimize the number of federated schema needed, as well as their maintenance. They also allow easier solutions to other problems.

1. Introduction

When some users want to operate jointly with several pre-existing, autonomous databases, a number of problems arise. Not only these databases may be in different sites, on different platforms, managed by different DBMSs, and using different data models, but moreover they were designed independently, and they are administered, maintained, and operated autonomously. Heterogeneity and autonomy are thence the main characteristics of such an Interoperable or Federated Database System (FDBS).

The syntactic heterogeneity implied by the different data models is usually overcome by adopting a common, canonical model and converting each local schema (component schema) into it. We will use in this paper the concepts and terms of the five level architecture of [SheLar90]. These component schemas, filtered into export schemas, are then associated into one or more federated schemas, resolving semantic heterogeneities between them. Finally, external schemas are derived at the top.

FDBS can be categorized as loosely coupled or tightly coupled based on who associates the component DBs into a FDBS: the users, or a Federation DBA.

Different users or classes of users may have different semantic interpretations of the data in the Component DBs. The solution in [SheLar90] is to construct a different federated schema for each of them. We will see that using *discriminated operations* a single federated schema can support multiple semantics.

This paper is organized as follows. In section 2 we introduce the Federation Rules to be followed when associating export schemas into federated schemas. In section 3 discriminated operations are introduced, particularly in object oriented and relational models. Section 4 explains how these discriminated operations may be used for FDBS, and the problems they solve or alleviate. In section 5 other uses for these operations are showed. We conclude in Section 6.

2. Federation rules for interoperable databases

FDBS are in a way the contrary of Distributed Database Systems (DDBS). In the latter case, a global schema is distributed into a number of local schemas; this is a top down, analytic approach. In FDBS, a number of preexisting local schemas (in fact via component and export schemas) are associated into one or more federated schemas; it is a bottom up, synthetic approach.

In DDBS, and following [CerPel84], the global schema is first fragmented, then allocated (each fragment to one or more nodes), and finally (each node) assigned to a specific DBMS. It is desirable to have transparencies (independence) from local DBMS, from location, and from fragmentation.

In FDBS, and following [SheLar90], each local schema is converted into a component schema in a canonical model, each component schema gives place to one or more export schemas, then these are associated by "constructing processors" to form federated schemas, and finally external schemas are derived from these. It is not necessarily desirable to have transparency from the component schemas; on the contrary, many applications will want to know the source of each piece of data. For a justification of this need, from an information systems point of view, see [WanMad90].

In DDBS, fragmentation must obey some rules to work properly:

- **completeness:** all data in the global DB (in each relation, if the model is relational) must be present in some fragment;

- reconstruction:** it must be possible to build the global DB (each global relation) from its fragments; and
- disjointness:** each piece of information must be present in only one fragment.

In FDBS, we may ask whether there are, similarly, some rules to be followed when associating export schemas into federated schemas. We formulate three rules, counterparts to the fragmentation rules, and we will call them **Federation Rules**:

- completeness:** all data in an export DB must be present in the federated DB;
- decomposition:** any query against a federated schema must be decomposable into queries against component schemas (export schemas in a first step); and
- discrimination:** each data in a federated DB must be traceable to the component DB it comes from. In other words, one must be able to discriminate the federated DB into its different sources.

Note that the discrimination rule is the contrary of transparency. This discrimination will be filtered out in external schemas for users and applications not needing the source of the data, and will be kept in external schemas for applications and users wanting it.

3. Discriminated operations

3.1 Discriminated union of sets

In dealing with sets, it is often useful to define a new set, not just as the ordinary union of two known sets, but as the discriminated union of two sets. Each element in the result carries a *discriminant* (or tag) indicating from which of the original sets it comes from [DahDijHoa72]. The cardinality of the discriminated union is always the sum of the cardinalities of its constituent sets: no duplicates may exist, because of the discriminant.

3.2 Discriminated union in programming and database languages

This structure has been used in several programming languages, from Algol68 to C, where a type may be declared as the discriminated union of any two types; other languages such as Pascal or Ada, have some form of discriminated union (only within records: variants). To ensure type checking, access to the value of a variable of the discriminated union type must be done with an alternative construct such as case [GheJaz82]. Also in [Third-Gen90] a rich type system is proposed (proposition 3.1.1), where one of the desired constructors is the union type that allows to construct a data element which can take a value from one of several types.

3.3 Discriminated Generalization in Object Oriented Models

Semantic and Object Oriented data models usually define some form of generalization, as in [Codd79, SmiSmi77, PecMar88]. When generalizing several (sub)classes into a super-

class, as is needed for FDBS, a discriminated form of generalization is useful. Each object of a subclass, when viewed as member of the superclass, has a *discriminant attribute*, whose value is, for example, the name of its subclass.

3.4 Discriminated union in the Relational Model

Union is a set operator that is constrained to apply only to pairs of union-compatible relations and so is the Discriminated Union. But the result of this last operation has not the same schema as its operands, because it has an additional attribute, the *discriminant*. Let us assume that, by default, the values of the discriminant are the names of the operand relations. Relational languages, and SQL in particular, could be extended with this operation.

Note the difference between the discriminated union and the UNION ALL of the SQL language. The latter allows duplicates in the result, against relational theory. Discriminated union, on the contrary, produce no duplicate tuples and is conformant with the theory.

The discriminated union may be defined in terms of other operations, including ordinary nondiscriminated union, if there exists an operation that returns the name of its operand, as the NOTE operation in RM/T [Codd79]. Then, the discriminated union (DU) of R and S would be:

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

or, using the TAG op. of RM/T ($\text{TAG}(R) = R \times \{\text{NOTE}(R)\}$)

$$R \text{ DU } S = \text{TAG}(R) \cup \text{TAG}(S)$$

The operands of these UNION (U) operation are union compatible if and only if R and S are union compatible.

Note that the discriminated union has an inverse, the selection (restriction) operation, while the nondiscriminated union has no direct inverse.

3.5 Discriminated Outer Union in the Relational Model

This operator may be applied to any pair of relations, even if they are not (discriminated) union compatible. It is to ordinary outer union what discriminated union is to non discriminated union: its result has an additional attribute, the *discriminant*.

4. Uses of discriminated operations for FDBS

Discriminated operations allow us to satisfy the third of the Federation Rules, i.e. discrimination. In this context, the values of the discrimination attribute would be the names of the component DBs. Note that no discriminated version of join operations is needed.

A number of problems are better solved with these operations. In this paper we only outline four of them, using examples found in the research literature.

4.1 Let us use an example from [SheLar90] to show the advantages of our approach when dealing with multiple semantics. Suppose that there are two export schemas, each containing the entity SHOE. The possible colors of SHOE in component

schema1 are brown, tan, cream, white and black and in component schema2 are brown, tan, white and black. Users defining *different* federated schemas may establish different mappings, for example:

```

user1 maps: cream → cream in schema1 and
            cream → tan in schema2
user2 maps: cream → tan | cream in schema1 and
            cream → tan | white in schema2

```

What we propose is to define *just one* federated schema for a set of predefined different mappings. The materialized extension of the federated schema would be the discriminated-union of the component DB's, i.e. compon_db1.shoe and compon_db2.shoe. Then alternative views can be defined in the external schemas using the information found in the predefined mappings and in the federated schema. In every moment we can know the origin of any data item.

With our approach the semantics are not in the federated schemas and hence we gain independence with respect to changes in the schemas, having a true separation between federated and external schemas. So we deal with multiple semantics without the update problem. As opposite to this, in [SheLar90] *every* external schema has to be modified when the semantics of its federated schema change.

4.2 Many systems support generalization [KauDroNeu90, FanLitNeuSch88, Su86]. With the discriminated generalization, homogeneous treatment of disjoint and overlapping union is possible. For the next example we take Oil_Plant and Coal_Plant classes used by [SchNeu88] and add two instances to each class:

<pre> class Oil_Plant attrib: PlantName Produced: MWh OilFired: BarrelOfOil methods: FireOn PowerOff FillOil </pre>	<pre> class Coal_Plant attrib: PlantName Produced: MWh Consumed: TonOfCoal methods: Start PowerOff PutCoal </pre>
---	---

<pre> object Oil_Plant1 Oil_Plant2 inst_of: Oil_Plant Oil_Plant attr: PlantName:"oil1" "oil2" Produced: 0 80 OilFired: 10 58 </pre>	<pre> object Coal_Plant1 Coal_Plant2 inst_of: Coal_Plant Coal_Plant attr: PlantName:"coal1" "coal2" Produced: 30 0 Consumed: 5 1 </pre>
--	--

To integrate these two classes, we need to solve in the first place the name and scale differences without modifying them (to preserve autonomy), so we define the super-class:

```

class Power_Plant
  discrim_generalization_of: oil_plant, coal_plant [with discr]
  attr: Consumed: MJoule
  case oil_plant: OilFired * 3.5
  case coal_plant: Consumed * 1.2
  methods: PowerOn
  case oil_plant: FireOn
  case coal_plant: Start

```

Now, suppose we want to start idle plants. Just sending the message [idles PowerOn], where idles=[Power_Plant where Produced=0]

will generate two messages:

```

when idles = "Coal_Plant.Coal_Plant2" →
  ["Coal_Plant.Coal_Plant2" Start]
when idles = "Oil_Plant.Oil_Plant1" →
  ["Oil_Plant.Oil_Plant1" FireOn]

```

This was easy because there were not name and scale differences. Now we are asked to turn off those plants that have consumed more than or equal to 50 MJoules. Before sending the appropriate message(s) we want to know which plants satisfy the condition stated above:

X : [(Power_Plant where Consumed ≥ 50)]

that is transformed to:

Xo : [(Oil_Plant where (OilFired * 3.5 ≥ 50))]

Xc : [(Coal_Plant where (Consumed * 1.2 ≥ 50))]

now, operating the discriminated union between Xo and Xc: X = Xo DU Xc we can send the message [X PowerOff] where X have the discriminated values that trigger the appropriate subclass methods. Only one instance in this example satisfies the condition, when

```

X = "Oil_Plant.Oil_Plant2" →
  ["Oil_Plant.Oil_Plant2" PowerOff]

```

4.3 In [KauDroNeu90] the CsMonograph class is defined, which in turn is sub-class of Library. CsMonograph collect all physical copies of monographs at two libraries using, as no copies can reside at two different places at the same time, a disjoint union (category generalization). Hence, a decision, use of disjoint or overlapping union, has to be made depending on what you want to model.

Erroneous design decisions are reduced with the discriminated generalization, being the system who manages disjoint as well as overlapping unions in a uniform way.

4.4 Let us present another example that is extensively described in [WanMad90]. This paper address the issue "where is the data from", a fact that is closely related to our research work. The example has three databases:

Alumni with ALUMNUS, CAREER and BUSINESS relations
Placement with STUDENTS, INTERVIEW and CORPORATION relations

Company with FIRM and FINANCE relations

They suppose also that syntactic and semantic (including the domain mismatch problem) schema integration has been made. An advantage of our approach is that we don't have to extend the Relational Model, except for our discriminated operation; neither we are tied to one model, as can be seen in this section).

Basically, we use the Outer Join (OJ), when necessary, for intra-database operations and the Discriminated Outer Union (DOU) for inter-database operations. So, for example, we can OJ Company.FIRM with Company . FINANCE first and then

DOU Alumni . BUSINESS with Placement . CORPORATION and with the outer-joined one.

To solve the domain mismatch problem we make an *horizontal coalesce*. For example, if we project Oname and State (suppose that are attributes of the generated relation in the last example) it is possible that "IBM" appears twice in Oname and its values in State were not equal ("null" and "NY"). We are developing a methodology to decide the right value. So our approach can also cope with the domain mismatch problem easier than [Mad et al.89]

5. Other uses of discriminated operations

Besides its use in the context of DB interoperability, discriminated operations, and in particular discriminated union, may have other uses in stand alone databases. We focus on relational DBMSs.

5.1 Use in applications

Some applications may take advantage of the existence of the discriminated union. For example, the union of relations:

SUPPLIES (SUPPLIER#, PART#, PRICE)

BIDS (SUPPLIER#, PART#, QUOTED_PRICE)

loses track of the origin of each tuple. A discriminated union would preserve this information in its result, such as:

SUPPLIES_AND_BIDS (DISCR, SUPPLIER#, PART#, PRICE)

where the key is DISCR, SUPPLIER# and PART#.

5.2 Use in view definition

The SQL standard does not allow the use of unions in the definition of views, and so do most DBMS. Apparently, this is because views which might involve a sort are avoided, and a union uses sort for duplicate elimination. Discriminated union does not involve duplicate elimination, and could therefore be easily implemented in view definitions by DBMS.

5.3 Use by the optimizer

When the DBMS optimizer finds a union operation, it cannot know whether or not there will be duplicates, and has to select an algorithm that performs their elimination, usually by sorting. If the system, and the language, support discriminated union, the optimizer knows that no duplicate elimination will be needed and can select a more efficient algorithm.

6. Related, future work and Conclusions

The only work we know that tags data in the federated DB with its source (and also with the intermediate DBs) is [Wan Mad90]. Our approach is simpler, but effective, in obeying the Federation Rule of discrimination, and *does not* need their extension to relational algebra. We are not restricted, moreover, to the relational model, but have shown its use in a more semantic model.

Many points remain to be solved to overcome all semantic heterogeneities, and we intend to pursue research in this area.

Acknowledgements

This work has been partially supported by the PRONTIC project TIC89/0303.

References

- [CerPel84] S. Ceri & G. Pelagatti: *Distrinuted Databases, Principles and Systems*. McGraw-Hill, 1984.
- [Codd79] E.Codd, Extending the DB relational model to capture more meaning, *ACM TODS*, Vol.4, No.4, Dec. 79, 397-434
- [DahDijHoa72] O.Dahl, E.Dijkstra & C.Hoare, *Structured Programming*, Ed. Academic Press, 1972 London.
- [FanLitNeuSch88] P.Fankhauser, W.Litwin, E.Neuhold & M.Schrefl, *Global view definition and multidatabase languages - two approaches to DB integration*, *EUTECO 88*, 1069-1081
- [GheJaz82] C.Ghezzi & M.Jazayeri, *Programming Language Concepts*, Ed. Jhon Wiley & Sons, 1982 USA.
- [KauDroNeu90] ViewSystem: integrating heterogeneous information bases by object oriented views, *IEEE 6th. Int. Conf. on Data Engineering*, February 1990, Los Angeles
- [Mad et al.89] S.Madnick, R.Wang, et. al, *CISL: Composing Answers from disparate information systems*, 1989 Workshop on Heretogeneous Databases, Chicago, December 11-13
- [PecMar88] J.Peckham & F.Maryanski, *Semantic Data Models*, *ACM Computing Surveys*, Vol.20, No.3, Sept. 88, 153-189
- [SchNeu88] M.Schrefl & E.Neuhold, *Object class definition by generalization using upward inheritance*, *IEEE 4th. Int. Conf. on Data Engineering*, February 1988, Los Angeles, 4-13
- [SheLar90] A. Sheth & J. Larson: *Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases*. *ACM Computing Surveys*, December 1990.
- [SmiSmi77] J.Smith & D.Smith, *DB abstractions: aggregation and generalization*, *ACM TODS*, Vol.2, No.2, June 77, 105-133
- [Su86] S.Su, *Modeling integrated manufacturing data with SAM**, *Computer* 19,1, 34-49
- [Third-Gen90] The Committee for Advanced DBMS Function, *Third - Generation DBS Manifesto*, *OODB DS-4 Conference proceedings*, july 1990, Windermere, UK. Also in *SIGMOD RECORD*, Vol.19, No.3, september 1990.
- [WanMad90] Y.Wang & S.Madnick, *A Polygen Model for heterogeneous DBS: the source tagging perspective*, *Proceedings 16th. VLDB, Brisbane-Australia*, August 1990, 519-538