ON OPERATIONS ALONG THE GENERALIZATION/SPECIALIZATION DIMENSION

Elena Rodríguez[1], Alberto Abelló[1], Marta Oliva[2], Fèlix Saltor[1], Cecilia Delgado[3], Eladio Garví[3], José Samos[3]

[1] Universitat Politècnica de Catalunya
{aabello, malena, saltor}@lsi.upc.es
[2] Universitat de Lleida
oliva@eup.udl.es
[3] Universidad de Granada
{cdelgado, egarvi, jsamos}@ugr.es

Abstract. The need to derive a database schema from one or more existing schemas arises in Federated Database Systems as well as in other contexts. Operations used for this purpose include conforming operations, which change the form of a schema. In this paper we present a systematic approach to establish a set of primitive conforming operations that operate along the Generalization/Specialization dimension in the context of Object-Oriented schemas.

1 Introduction

Users of a DataBase (DB) need to retrieve and update its data, and its DB Management System (DBMS) must provide operations for that. Data in a DB are instances of a DB Schema, or *schema* for short, and operations on schemas are also needed. The topic of operations on schemas has been less studied than the one of operations on data.

A schema always follows a particular data model (for example, a Relational schema is a set of Relation schemas and Integrity Constraints (ICs) following the Relational data model). If a schema follows one of the multiple Object-Oriented (O-O) data models, then we have an O-O schema. The topic of operations on O-O schemas has been less studied than the topic of operations on Relational schemas.

In any system having a *multi-schema architecture*, with schemas at different levels, *mappings* among the schemas appear. An example is the old 3-level ANSI/X3/SPARC schema architecture framework. Other examples are Sheth & Larson's 5-level schema architecture for *Federated DB Systems* (FDBS) of [18], or our 7-level schema architecture of [13], with mappings such as those from Component Schemas to Export Schemas.

A system with a multi-schema architecture can be built by starting from one or more schemas, and applying operations to yield another schema (and their corresponding mappings), and so on. At each step, if all schemas follow the same data model, then the operations applied are precisely the schema operations of that model; this is the reason why it is important to use a data model with adequate schema operations.

In this paper we discuss schema operations needed to *conform* O-O schemas, i.e. to change the *form* of a given schema into a desired form. These conforming operations appear in several contexts, but our emphasis here is in the case of a FDBS using an O-O model as its *Canonical Data Model* (CDM) –in the sense of [18]–, particularly when constructing Federated Schemas from Export Schemas. This schema integration process can be considered as having two steps: first, each Export Schema is transformed into a common form, and then all these conformed schemas are integrated by some Generalization operation, as in [8].

We assume that these conforming operations do not add *semantic contents* to the obtained schema, because all *semantic enrichment* takes place when producing Component Schemas, as was presented in [6]. This is so because of the "separation of concerns" of the architectures. For the same reason conforming operations do not change the data model; translating operations do that.

We will be using in this paper the BLOOM99 data model of [2], but our results are applicable to other O-O data models as well.

Of the O-O "dimensions" present in many O-O data models, conforming operations in this paper work on the Generalization/Specialization (G/S from now on) dimension. Each operation produces a mapping between the input and the output schema, to be stored in a Directory; these mappings may be considered to lie along the Derivability dimension, and are out of the scope of this paper.

This paper is organized as follows: related work is covered in section 2, and the BLOOM99 data model in section 3; sections 4 and 5, the core of the paper, present our approach, and primitive conforming operations along the G/S dimension, respectively; conclusions, acknowledgements and references close it.

2 Related Work

As pointed out in the introduction, this paper focuses on Schema Conformation. This problem is related to other topics such as Schema Evolution and View Definition (definition of External Schemas and Derived Classes). The main difference between them is in their aims. The Schema Evolution objective is to modify a schema to adapt it to changes in the modeled Universe of Discourse (UoD); on account of this, these changes can add semantic contents. In the case of View Definition, its general objective is to transform and to present information stored in DB, all or part of it, according to end-user requirements. Besides our above mentioned motivation, the operations presented in the following sections can also be used to make schemas evolve or to define views, but these applications are out of the scope of this paper.

Next, we briefly review some of other authors' proposals on modification and transformation of schemas related to Schema Conformation, View Definition and Schema Evolution, both in O-O models, in subsection 2.1. Then, in subsection 2.2, we characterize our approach.

2.1 Operations on Object-Oriented Data Models

Directly related to the Schema Conformation problem is the proposal found in [11], where a set of operations to build and modify the generalization hierarchy is presented. In [9] generalization hierarchies can be merged using a set of operations; before merging hierarchies, relationships between attributes (Attribute Assertions) and between types (Generalization Assertions) have to be obtained to guide the process. In [10] some schema transformation operations are presented. One of these is the meet operation that restructures a schema containing two classes with a common key and overlapping attributes using generalization. In the transformed schema, all common attributes are removed from the original classes and a superclass with these attributes is created.

Several view mechanisms have been proposed for O-O models, as can be seen in [12]. In [14] proposal, Derived Classes are defined using Object Algebra, and then integrated in a global O-O schema to define External Schemas as subschemas of it. In [4], Derived Classes that include non-derived attributes (Capacity Augmenting Classes) can be defined using a specific language. In [17] and [16] different External Schema definition methodologies are proposed; in these, Derived Classes can contain existing or new objects (defined with Object Preserving or Object Generating semantics respectively). In [19] some examples of schema changes that do not add semantic contents, and can be simulated by views, are presented. In [15] a Schema Evolution mechanism, without limitation on the accepted changes, based on a View Definition system is proposed.

In the Schema Evolution field, one of the first proposals for O-O DBs can be found in [3]. In it, a taxonomy of primitive operations for the Orion system is defined; each operation has a semantic based on a set of rules that preserve Schema Invariants. The corresponding proposal for the O_2 system can be found in [20], in this case, classes and types can be defined in the O-O model, and primitive operations are defined according to this fact. In [7] a framework is proposed based on the Object Data Management Group (ODMG) data model; in it, complex schema evolution operations can be flexibly defined as a sequence of primitive ones defined in the underlying ODMG system.

2.2 Our Approach

In this paper, a systematic approach to establish a set of primitive conformation operations along the G/S dimension in an O-O model is carried out. One difference between our approach and the previously mentioned proposals is that in these O-O models the G/S dimension is not explicitly present because they do not establish any criterion for the specialization. Another difference is that in our operations the schema form is taken explicitly into account; thus, from a starting schema, according to its patterns, operations can be applied to obtain a conformed schema to be integrated lately.

3 The BLOOM99 Data Model

The BarceLona O-O Model (BLOOM) was conceived to act as CDM in the integration process of a FDBS. [2] explains the revised version of the model, while figure 1 shows its main metaclasses plus those metaclasses belonging to the G/S dimension (using BLOOM99 syntax in [1], see the legend). It can be seen that three different kinds of abstractions are considered i.e. *classes*, *nexus*, and *functions*:

- Classes represent sets of objects sharing structure and semantics. All class instances will have the same set of attributes, and will represent the same kind of entity in reality. Every class is identified by *name*, and composed by a predicate its instances fulfill. This predicate is aggregated as *intension* of the class.
- Nexus indicate relationships among classes. Every nexus is composed by a destination class (we call it *to*), a set of origin classes (we call it *from*), and a set of functions with a different meaning depending on the kind of nexus (we call it *tag*).
- Functions represent knowledge about objects in the UoD. For instance, person(x) would be a boolean function indicating whether an entity is a person or not. A function is identified by its *name*, as well.

All_nexus is specialized attending to the O-O dimension it belongs to. In this case, since we are only considering those nexus in the G/S and Aggregation/Decomposition (A/D) dimensions, the specialization is non-complementary. This specialization gives rise to *Specialization* and *Aggregating* metaclasses. Given that we deal with conforming operations along G/S dimension, only metaclasses associated to this dimension are presented afterwards. For information about A/D dimension see [2].

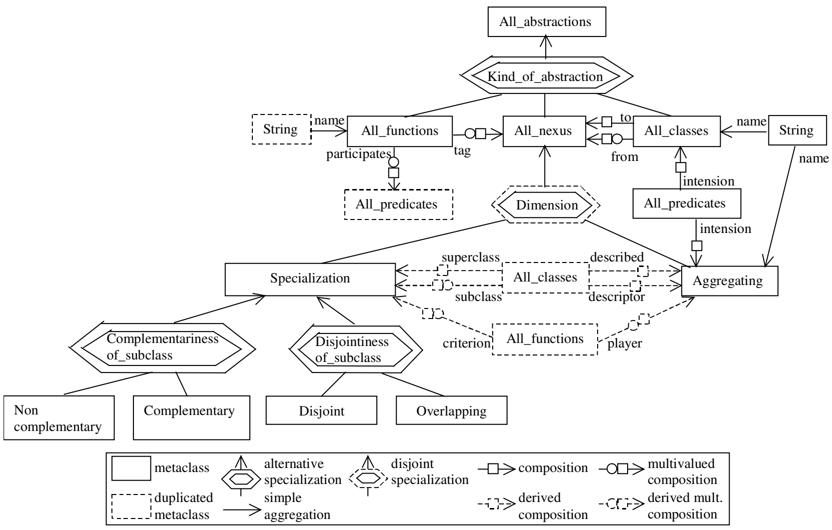

Fig. 1. BLOOM metaclasses

The *Specialization* metaclass is specialized depending on its complementariness and disjointness, giving rise to four different kinds of specialization corresponding to the cartesian product of the subclasses obtained at both specializations of the *Specialization* metaclass: *complementary* (i.e. *Complementary*×*Overlapping*), *disjoint* (i.e. *NonComplementary*×*Disjoint*), *alternative (*i.e. *Complementary*×*Disjoint*), and *general* (i.e. *NonComplementary*×*Overlapping*). A specialization being complementary means every instance of the superclass is, at least, in a subclass. A disjoint specialization implies that every instance of the superclass is just allowed to be, at most, in a subclass. General means it is neither disjoint, nor complementary, while alternative means being complementary and disjoint at the same time.

Since *Specialization* is subclass of *All_nexus*, it is also composed by *from*, *to*, and *tag* attributes. These attributes are renamed to be *subclass*, *superclass*, and *criterion* respectively. Thus, a specialization nexus connects subclasses with a superclass based on a given set of functions used as criterion. Those functions are used to restrict the presence of instances of the superclass in each subclass. For instance, class `Person` could have predicate "`person(x)=true`" as intension. Thus, if it is specialized by age, the intension of its subclass `Underage` could be "`person(x)=true and agePerson(x)<18`". The criterion of a specialization is the set of functions added to the predicates of the subclasses w.r.t. that of the superclass.

Notice that specializations are absolutely independent of the information represented along the A/D dimension. We only know that an instance belongs to a subclass because it fulfills the intension predicate. Over our example, an attribute `age` does not necessarily exist in the class `Person`, so that if the subclass `Underage` disappears, we can not recover its information.

The specialization digraph is a semi-lattice, having class *All_objects* as root. It is not a tree, because more than one specialization path is allowed between two classes. Nevertheless, there is a really strong IC on this allowance: both paths must involve the same set of functions, however, it is not possible that given any two classes (one in each path) both sets of functions used in the specializations till then coincide.

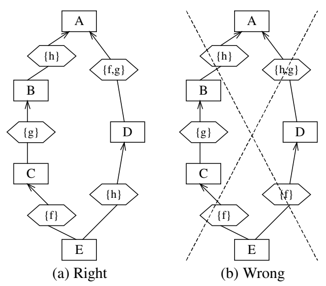

Fig. 2. Specialization integrity constraint

Figure 2 shows a correct and an incorrect example. In 2.b, classes `C` and `D` should coincide, since functions used to specialize them (i.e. `{h,g}`) coincide. They are superclass of the same class by the same criterion, thus they should be the same class.

4 Choosing Schema Patterns and Operations

The main difference among operations defined in this work and those defined by other authors is that we only consider operations modifying the form of the schema. For example, operations that change names (class name, function name, etc.) are not considered. Since they do not change any form, they are not needed to conform schemas.

Operations have a *source schema* as input, and produce a *target schema* as output. They locate a particular pattern in the source schema, and change it to a different form in the target schema. Subsection 4.1 identifies patterns of interest along the G/S dimension, while 4.2 explains some properties of the set of chosen operations.

4.1 Patterns of interest along the Generalization/Specialization dimension

We define a *pattern of interest* as that which represents the minimal information needed to apply a given conforming operation. As candidate patterns, we took digraphs, where nodes represent classes, and arcs represent specialization nexus between classes.

To avoid considering the same candidate pattern twice, we took connected *digraph equivalence classes* (DECs) defined by an isomorphism as in [5]. Moreover, some equivalence classes were not taken into account due to the following hypothesis:

1. New semantics cannot be added to the schema. Therefore, we only consider connected DECs, so that if we have a disconnected graph, operations will separately work in each of the components.
2. It is not necessary to study multidigraphs, i.e. digraphs with more than one arc between two nodes in the same direction, since in the specialization dimension they represent the same specialization nexus, and thus one of them is redundant.
3. DECs containing cycles do not need to be considered, since cycles are not allowed in the G/S dimension.

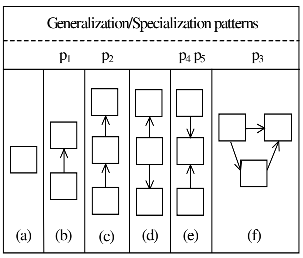

Fig. 3. Considered equivalence classes

As consequence of the previous hypothesis, it was found that it was only necessary to consider six DECs. We do not consider patterns having more than three classes because we did not find any operation along G/S over them that cannot be obtained by composition of operations over patterns with less than four classes. Thus, let us analyze each DEC (from (a) to (f) in figure 3), and identify how they could be used as a pattern in the G/S dimension:

(a) This DEC represents a class without nexus, and it cannot be considered as a source for any operation along the G/S dimension. Given that operations do not add semantic contents, adding new specialization nexus or new subclasses it just possible if they are derived from other existing specialization nexus, or other existing subclasses.
(b) This gives rise to pattern p_1 and it represents a class that has been specialized into a subclass.
(c) This DEC gives rise to a pattern p_2 and it represents a serial G/S hierarchy.
(d) In the G/S dimension, these two nexus cannot be considered alone, since there is an IC forcing to have a top semi-lattice, so it does not produce any pattern.
(e) In the G/S dimension, this DEC can be interpreted as either two independent specializations, or one specialization into two subclasses. Therefore, there are two patterns associated to DEC (e), i.e. p_4, and p_5.
(f) DEC (f) provides an interesting pattern for G/S, p_3, since it gives rise to a semi-lattice.

Therefore, we have found five patterns of interest, whose transitions are studied in next section.

4.2. Identifying Primitive Operations along the G/S Dimension

A schema can be seen as composed by different patterns. Thus, the conformation of a schema can be understood as a *composition of operations*, which is associative, but not commutative. We call *primitives* to a set of atomic operations needed to conform schemas, while *derived* operations are those that can be obtained by composition of primitives. In the G/S dimension, we have established three different sets of primitive operations. For each set, its operations are primitives in the sense that none of them can be left out without affecting the set of derived operations.

For every different set of primitive operations, patterns of interest and primitive operations can be represented in a digraph $G_k = (P, O_k)$, where P is the set of nodes that represents patterns, and O_k is the set of arcs that represent primitive operations chosen in option k (k=1…3). In a given G_k, an arc o_{ij} from p_i to p_j represents the primitive operation that converts pattern p_i into pattern p_j. Figure 4 shows the pattern transition graph associated to each option.

Although cardinality of O_1 is different to cardinality of O_2 and O_3, each set O_k is minimal, in the sense that for any operation o_{ij} belonging to O_k, o_{ij} cannot be derived from the set of operations given by $O_k - \{o_{ij}\}$.

Mathematically, every G_k is complete and connect, since any of them contains an eulerian circuit. Furthermore, G_1 is a symmetric digraph, in the sense that for every arc o_{ij}, there is an arc o_{ji}; o_{ji} represents the *opposite* operation, which may or may not be the mathematical *inverse* of o_{ij}. When o_{ij} looses semantic contents, o_{ji} works on the contrary sense of o_{ij} giving rise to a final target schema different to the initial source schema, because o_{ji} is not able to add the information disregarded by o_{ij}. Therefore, we can just affirm that the patterns will have the same form. If some information was lost an opposite operation does not recover it. On the other hand, G_2 and G_3 are not symmetric, given that for every arc o_{ij}, do not always an arc o_{ji} exists. That means that in G_2 and G_3 there are opposite operations that are not primitive, but derived.

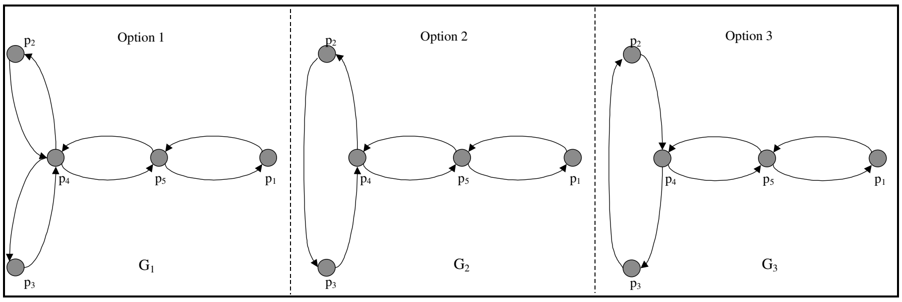

Fig. 4. Patterns transition digraphs

Every digraph G_k could also be represented as an adjacency sparse matrix M of size 5x5, containing eight "1's" in case of G_1, and seven "1's" in case of G_2 and G_3. In figure 5, we show the primitive operations over the patterns of interest that have been identified for every G_k collapsed in just one matrix, although they should have been spat in three different matrixes. Cells of the matrix containing just the operation name correspond to operations considered in all digraphs, while operations containing the additional tags 'G_k' correspond to primitive operations that should be just taken into account in digraph G_k. It is important to note that this collapsed matrix is not a proper adjacency matrix, since it contains the names of the operations corresponding to the arcs instead of just "1's".

Empty cells of the matrix do not mean that the transition from a source pattern to a target pattern is not allowed. The operations to be placed in those cells correspond to derived operations. Notice that the operation located at cell c_{ij}, is the opposite operation of that in the symmetric cell c_{ji}.

5 Operations along the Generalization/Specialization Dimension

Instead of presenting all possible sets of conforming operations for each digraph G_k, in this section we concentrate on the set of conforming operations corresponding to G_3. This choice has been made on the basis that we believe that G_3 represents, jointly with G_2, most intuitive transitions between patterns than G_1, and this fact facilitates the comprehension of conforming operations. Subsection 5.1 explains primitive operations on G_3, while subsection 5.2 shows some interesting derived operations of G_3. It is important to note that all operations are applied on BLOOM99 schemas, although they are also applicable to other O-O data models as well.

5.1 Primitive Operations

In G_3 there are seven primitive conforming operations that work exclusively along the G/S dimension. All operations have an opposite, which coincides with the inverse, except in the case of *EliminatingSubclass* because, as we will see later, this operation produces a schema with poorer semantic contents. Notice also that some inverse operations are derived.

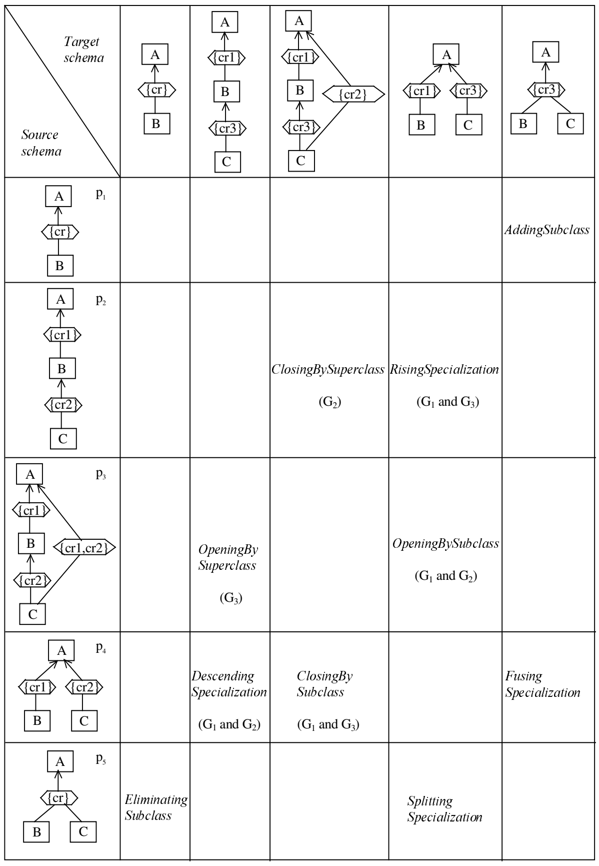

Fig. 5. G/S dimension collapsed matrix

The *AddingSubclass* operation, located at cell c_{15} in the matrix shown in figure 5, adds a new subclass to an existing specialization. Although it seems that this operation incorporates new information to the schema, this is not the case. Actually, it obtains a new subclass reorganizing preexisting subclasses (i.e. union, intersection, difference, etc. of their instances) without changing the criterion of the specialization. Generally, this operation causes a change in the kind of specialization.

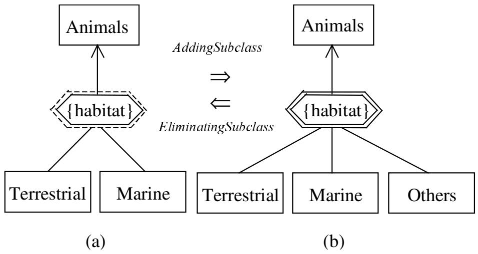

Fig. 6. *Adding/EliminatingSubclass* example

We can see in figure 6.a a disjoint specialization, using the criterion habitat, of class `Animals` into subclasses `Terrestrial` and `Marine`. Figure 6.b shows how we add a new subclass `Others` to the previous disjoint specialization. The instances of `Others` are determined by difference between instances of `Animals` and the union of instances of `Terrestrial` and `Marine`. Thus, the kind of the specialization changes from disjoint to alternative.

With the use of the *EliminatingSubclass* operation (cell c_{51}) we can remove a subclass from a specialization. From the source schema in figure 6.b, the schema 6.a is obtained by eliminating the subclass `Others`. While *EliminatingSubclass* is always the inverse operation of *AddingSubclass* (we can add and remove several times `Others`), *AddingSubclass* is just the opposite of *EliminatingSubclass*. There are some cases where it is impossible to recover an eliminated subclass without adding additional information. For instance, if we eliminate `Marine` from the schema depicted in figure 6.a, we cannot recover the previous schema because the intension of subclass `Marine` and the information related to its instances have disappeared. Note again that we do not assume the existence of an attribute `habitat` in class `Animals`, and if this were the case, the operation that could recover the eliminated subclass `Marine` would work simultaneously along the G/S dimension as well as the A/D dimension.

Notice that, as it can be seen in the matrix, the *EliminatingSubclass* operation is only applicable if the specialization has, at least, two subclasses. As we will see in the subsection 5.2, the operation that eliminates a whole specialization is a derived operation. The decision is justified by the fact that we consider that every primitive operation performs just one atomic action from a conceptual point of view, and the elimination of the last subclass of a specialization implies not only the elimination of the subclass, but also the elimination of the entire specialization.

The *RisingSpecialization* operation (cell c_{24}) derives a new specialization of a superclass by applying the transitivity property. In other words, we can see a subclass of a subclass of a superclass just as a direct subclass of the superclass. In figure 7, we can see an example of *RisingSpecialization*, where we obtain the subclasses `Marine Mammal` and `Terrestrial Mammal` (see figure 7.a) as direct subclasses of superclass `Animals` (see figure 7.b). Figure 7.b also shows how specialization criterion is modified to contain all those used in the path from the superclass to the subclasses in the source schema.

ClosingBySubclass (cell c_{43}) closes a specialization path between two classes. As consequence of this operation, two specialization paths between two classes have been generated giving rise to a semi-lattice. It is important to notice that, in general, it is not applicable, since there is a strong IC on this (see section 2). In figure 7 we can see an example of *ClosingBySubclass*, which shows subclasses `Marine Mammal` and `Terrestrial Mammal` not only as subclasses of superclass `Animals` (see figure 7.b), but also as subclasses of subclass `Mammal` (figure 7.c).

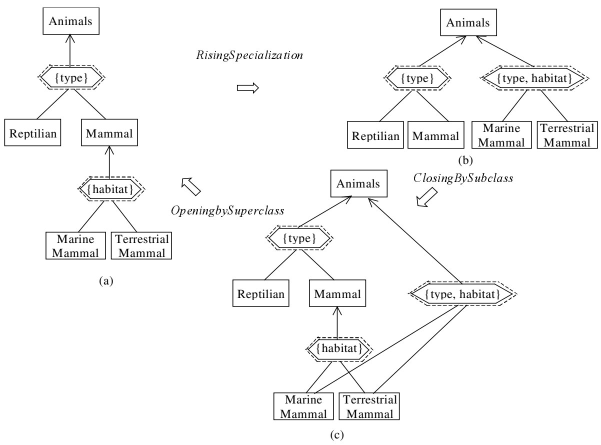

Fig. 7. *RisingSpecialization, ClosingBySubclass and OpeningBySuperclass* example

The *OpeningBySuperclass* operation (cell c_{32}) changes a semi-lattice pattern into a tree pattern by removing part of one path between two classes (i.e. a specialization), eliminating multiple inheritance. This does not imply any information loss because, as stated in section 3.1, when there are two specialization paths between two classes, both involve the same set of criteria. In the example in figure 7, we can see how the specialization with `{type, habitat}` criterion disappears. So that schema 7.c changes into 7.a. Now, `Marine Mammal` and `Terrestrial Mammal` are just subclasses of a unique superclass, in this case `Mammal`.

The *FusingSpecialization* operation (cell c_{45}) fuses all subclasses in two different specializations of a superclass into a unique specialization of that superclass. The generated specialization has as criterion the union of criteria of the source specializations. Its inverse, *SplittingSpecialization* (cell c_{54}), splits subclasses of a specialization of a class into two different subsets forming different specializations. In this case, the criterion of each specialization is a subset of the criterion of the specialization of source schema. At the same time, the distribution of subclasses between specializations in the target schema has to be decided. These operations are more frequently used as intermediate operations than to obtain a definitive target schema.

Figure 8 shows how both disjoint specializations of `Animals` in 8.a are fused into a unique general specialization using criteria (`{type, habitat}`) of previous specialization. `Reptilian`, `Mammal` and `Marine` that were subclasses by both previous specializations are now subclasses by the new unique specialization.

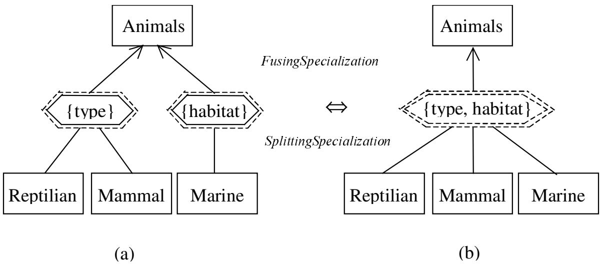

Fig. 8. *Fusing/SplittingSpecialization* example

5.2. Derived Operations

Due to space limitations, we only present some interesting operations associated to G_3: *DescendingSpecialization*, *EliminatingSpecialization* and *CartesianProduct*.

The *DescendingSpecialization* transforms pattern p_4 into p_2. It is derived by means of *ClosinBySubclass* and *OpeningBySuperclass*, and it is the inverse operation of *RisingSpecialization*. Descending a specialization requires that its criterion include the criterion of the specialization of the subclass where we want to place it, due to the usage of *ClosingBySubclass*. So, whereas sometimes it is possible to directly apply *DescendingSpecialization*, in other cases it is necessary to previously use some operations to obtain an adequate pattern (i.e. *FusingSpecializatio*n, *AddingSubclass*, and *SplittingSpecialization*). Figure 7 shows the effect of *DescendingSpecialization* when it is applied to the source schema represented in 7.b producing the target schema depicted in 7.a.

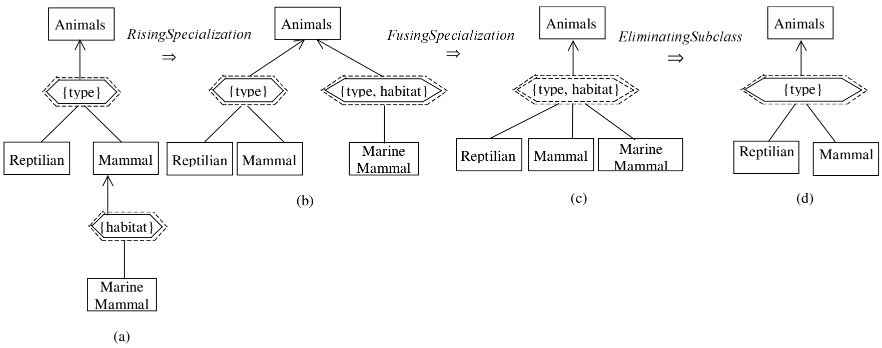

Fig. 9. *EliminatingSpecialization* example

The *EliminatingSpecialization* (to be placed at cell c_{21}) transforms pattern p_2 into pattern p_1. It is obtained by the sequence of *RisingSpecialization*, *FusingSpecialization* and *EliminatingSubclass*. Figure 9 shows the effect of this operation by eliminating the specialization by `{habitat}` from the source schema presented in 9.a giving rise to a final target schema represented in 9.d.

The *CartesianProduct* of specializations changes pattern p_4 into p_5, and it should be placed in cell c_{45}. Let us apply *CartesianProduct* to the two specializations of `Animals` shown in figure 10.a to obtain figure 10.b. It is derived by the next sequence of primitive operations: apply first the *FusingSpecialization*; then apply four times the *AddingSubclass*, once for each of the subclasses that appear in figure 10.b, by intersecting the instances of each subclass of one specialization with the instances of each subclass of the other one; finally, get rid of initial subclasses by applying *EliminatingSubclass*.

In general, there is a loss of information, and therefore no inverse operation exists. Note that we are not assuming that `Animals` has attributes `type` and `habitat`, that would lie along the A/D dimension; we are working solely on the G/S dimension. Only if both original specializations were complementary, the resulting specialization would also be complementary, and there would be an inverse operation.

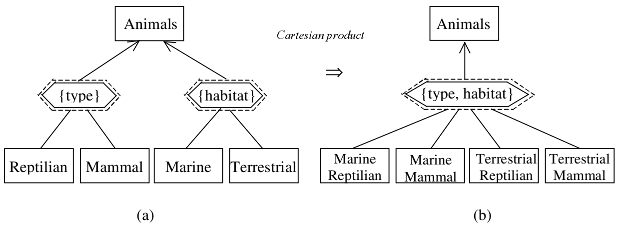

Fig. 10. *CartesianProduct* example

6 Conclusions

Conforming operations change a pattern in a source schema into a different pattern in a target schema, changing the form of the schema, but not adding semantics. We have presented how a systematic approach that explicitly takes into account the G/S dimension, determines the set of all patterns of interest, and establishes possible sets of primitive conforming operations. Other conforming operations can be derived from these primitives.

Current work includes the application of the presented approach to the A/D dimension. This allows not only the identification of conforming operations that exclusively work along the A/D dimension, but also the derivation of operations that work simultaneously along the G/S dimension as well as the A/D dimension.

Finally, each operation produces a mapping between its source and its target schema. These mappings lie along the Derivability dimension, which constitutes our future work.

Acknowledgments

This work has been partially supported by the Spanish Research Program PRONTIC under projects TIC2000-1723-C02-(01 and 02), as well as grant 1998FI-00228 from the Generalitat de Catalunya.

References

1. A. Abelló, M. Oliva, E. Rodríguez, F. Saltor. *The Syntax of BLOOM99 Schemas*. Technical Report LSI-99-34-R. Dept LSI, Jul. 1999.
2. A. Abelló, E. Rodríguez: Describing BLOOM99 with regard to UML Semantics. In *JISBD'2000*, Valladolid (Spain), Nov. 2000, pp. 307-319.
3. J.Banerjee, W.Kim, H.J.Kim,H. F.Korth: Semantics and Implementation of Schema Evolution in Object-Oriented Databases.*ACM SIGMOD Conference*, 1987, pp. 311-322.
4. E. Bertino: A View Mechanism for Object-Oriented Databases. In *Advances in Database Technology -. EDBT'92*, LNCS 580, Springer. 1992, pp. 136-151.
5. J. Bradley. *Introduction to Discrete Mathematics*. Addison-Wesley, 1988, pp. 469-524.
6. M. Castellanos, F. Saltor, M. García-Solaco: Semantically Enriching Relational Databases into an Object Oriented Semantic Model. In: D. Karagiannis (ed.): *Database and Expert Systems Applications* (DEXA'94). Springer, LNCS 856, 1994, pp. 125-134.
7. K. T. Claypool, J. Jing, E. A. Rundensteiner: SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework. In *Proc. 1998 Int'l Conf. on Information and Knowledge Management*, ACM 1998, pp. 314-321.
8. M. García-Solaco, F. Saltor, M. Castellanos: Semantic Heterogeneity in Multidatabase Systems. In O. Bukhres and A. Elmagarmid (eds.) *Object Oriented Multidatabase Systems.* Prentice-Hall, 1996.
9. M. Mannino, S. Navathe, W. Effelsberg: A Rule-Based Appoach for Merging Generalization Hierarchies. *Informations Systems 13*, (3), 1988, pp. 257-272.
10. R. J. Miller, Y. E. Ioannidis, R. Ramakrishnan: The Use of Information Capacity in Shema Integration and Translation. In *Proc. of the Int'l Conf. VLDB*, 1993, pp. 120-133.
11. A. Motro: Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering 13*, Jul. 1987, pp 785-798.
12. R. Motschnig-Pitrik: Requeriments and Comparison of View Mechanisms for Object-Oriented Databases. *Information Systems. 21*, (3), 1996, pp. 229-252.
13. E. Rodríguez, M. Oliva, F. Saltor, B. Campderrich: On Schema and Functional Architectures for Multilevel Secure and Multiuser Model Federated DB Systems. In *Proc. Int'l. CaiSE'97 Workshop* EFDBS'97, 1997, pp. 93-104.
14. E.A.Rundensteiner:MultiView:A Methodology for Supporting Multiple Views in Object-Oriented Databases.In *Proc. 18th VLDB Conf.*,Morgan Kaufmann, 1992,pp.187-198.
15. E. A. Rundensteiner, A. J. Lee,Y-G Ra: Capacity-Augmenting Schema Changes on Object-Oriented Databases: Towards Increased Interoperability. In *OOIS'98 Proc.,* Springer. 1998, pp 349-366.
16. J. Samos: Definition on External Schemas in Object Oriented Databases. In *Proc. OOIS'95*, Springer, 1995, pp. 154-166.
17. C. S. Santos, S. Abiteboul, C. Delobel: Virtual Schemas and Bases. In *Advances in Database Technology – EDBT'94*, LNCS 779, Springer, 1994, pp. 81-94.
18. A. Sheth, J. Larson: Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. *ACM Computing Surveys 22* (3), 1990, pp. 183-236.
19. M. Tresch, M. H. Scholl: Schema Transformation without Database Reorganization. *ACM SIGMOD Record 22* (1), 1993, pp. 21-27.
20. R. A. Zicari: A Framework for Schema Updates in an Object-Oriented Database System. In F. Bancilhon, C. Delobel and P. Kanellakis (eds) *Building an Object-Oriented Database System: The Story of O_2*, Morgan Kaufmann, 1992, pp. 146-182.