ON OPERATIONS TO CONFORM OBJECT-ORIENTED SCHEMAS

Alberto Abelló, Elena Rodríguez, Fèlix Saltor
Universitat Politècnica de Catalunya
Email: {aabello, malena, saltor}@lsi.upc.es

Marta Oliva
Universitat de Lleida
Email: oliva@eup.udl.es

Cecilia Delgado, Eladio Garví, José Samos
Universidad de Granada
Email: {cdelgado, egarvi, jsamos}@ugr.es

Keywords: Information Integration, Cooperative Information Systems, Federated Database Systems, Object-Oriented Schemas.

Abstract: To build a Cooperative Information System from several pre-existing heterogeneous systems, the schemas of these systems must be integrated. Operations used for this purpose include conforming operations, which change the form of a schema. In this paper, a set of primitive conforming operations for Object-Oriented schemas are presented. These operations are organized in matrixes according to the Object-Oriented dimensions -*Generalization/Specialization*, *Aggregation/Decomposition*- on which they operate.

1 INTRODUCTION

A "Cooperative Information System" (CIS) is built upon a number of pre-existing heterogeneous information systems. We assume that each one of them will have a schema in some data model (relational, object-oriented, ...). Because of the autonomy of design of these systems, their models will, in general, be different: "syntactic heterogeneity". One of the methods to overcome this heterogeneity is by adopting a data model as the "Canonical Data Model" (CDM) of the CIS, and translating schemas from their native models to the CDM, as explained in Sheth and Larson (1990). "Object-Oriented"(O-O) data models were found in Saltor *et al.* (1991) well suited as CDM of a CIS, and we will be assuming in this paper an O-O CDM.

The architecture of a CIS may follow different frameworks, but most likely it will be a "multilevel-schema architecture", with schemas at different levels, and schema *mappings* between consecutive levels. The now classical example is Sheth & Larson's 5-level schema architecture for "Federated Database Systems" (Sheth and Larson, 1990); it continues to be a useful reference framework (Conrad *et al.*, 1999).

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

This paper discusses schema operations needed to *conform* "Object-Oriented" (O-O) schemas, i.e. to change the *form* of a given schema into a desired form. Our emphasis here is in the process to build a CIS using an O-O model as its CDM, particularly when constructing *Federated Schemas* from *Export Schemas* in the sense of Sheth and Larson (1990). This schema integration process has two steps: first, each *Export Schema* is transformed into a common form (conformation) and then all these conformed schemas are integrated by some "Generalization" operation, as in García-Solaco *et al.* (1996).

Here, a set of *primitive* operations to *conform* O-O schemas are presented. In these 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. We assume that these conforming operations do not add new information (they do not augment the information capacity of the schema in the sense of Miller *et al.* (1993)) to the obtained schema (all elements added must be derived from the source schema). All "semantic enrichment" takes place when producing *Component Schemas*, as was presented in Castellanos *et al.* (1994). 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 use UML terminology (OMG, 2002). We add to this terminology some general integrity constraints to restrict the semantics of some operations. However our results are applicable to any O-O data model.

Conforming operations are defined on the *Generalization/Specialization* (G/S from now on) and on the *Aggregation/Decomposition* (A/D) dimensions. Each operation produces a mapping between the input and output schemas, and all these mappings will be stored in a repository; but this topic is out of the scope of this work.

This paper is organized as follows: the UML model elements are introduced in section 2; section 3 the schema patterns and *primitive* conforming operations are presented; related work, conclusions and references close it. We use **bold face** for UML terms, *italic* for new terms, and "quotes" for terms introduced by other authors.

2 MODEL ELEMENTS

Figure 1 shows the UML metaclasses meaningful for our purpose. We only consider three elements: **Class**, **Generalization**, and **Association**.

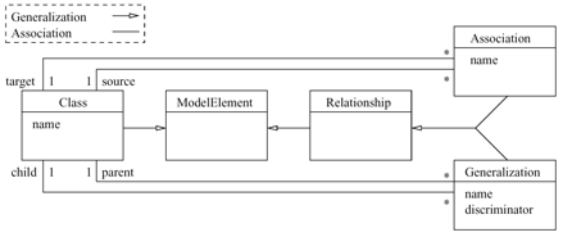

Figure 1. Subset of UML metaclasses.

- **Classes** describe sets of objects sharing structure and semantics, which can be given either extensionally, or intensionally.
- **Generalizations** are **Relationships** between two **Classes** along the G/S dimension.
- **Associations** are **Relationships** between two **Classes** along the A/D dimension.

2.1 Generalization Metaclass

A **Generalization** is a taxonomic relationship between a more general element (**parent**) and a more specific one (**child**). It uses a **discriminator** to show the criterion used in the specialization.

A set of **Generalizations** sharing the same **parent** and **discriminator** form a **Partition**. In our context, a **discriminator** is a function used to restrict the presence of instances of the superclass in each subclass. The **discriminator** of a **Generalization** is the set of functions added to the predicate of the **child** w.r.t. that of the **parent**.

Notice that **Generalizations** are absolutely independent of the information represented along the A/D dimension (which includes attributes, as explained below). We only know that an instance belongs to a subclass because it fulfils the intension predicate. Attributes corresponding to the **discriminator** do not necessarily exist, so that if the subclass is deleted, the information may be lost.

We assume that the generalization digraph is a semi-lattice, having **Class** *All_objects* as root. It is not a tree, because more than one generalization path is allowed between two **Classes**. Nevertheless, we impose a really strong Integrity Constraint (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 **Generalizations** till them coincide. Figure 2 shows a correct and an incorrect example. In (b), **Classes** `Girl` and `YoungWoman` should be the same, since functions used to specialize them (i.e. {`sex(x)`, `agePerson(x)`}) coincide. In addition, they are superclasses of the same **Class** (`GirlScout`) by the same **discriminator** (`hobby(x)`), thus they should be the same **Class**.

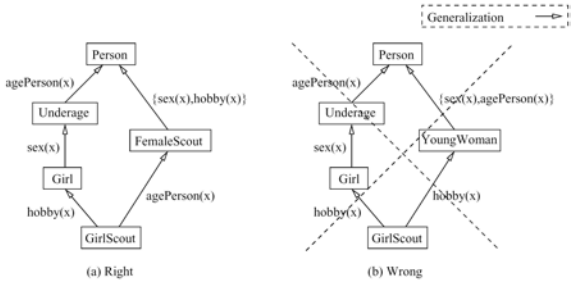

Figure 2. **Generalization** integrity constraint.

2.2 Association Metaclass

An **Association** defines a semantic relationship between **Classes**. An instance of an **Association** is a **Link**, which is a tuple of instances obtained from the related **Classes**. In general, UML allows n-ary **Associations**. However, we only consider binary unidirectional **Associations**. Having only binary **Associations** does not reduce the expressivity of an O-O schema, since n-ary **Associations** can always be represented by a **Class** and n binary **Associations**. UML refers to the **Association** ends as **source** and **target**; here, the **target** end is graphically expressed by the head of the arrow.

Moreover, as we did for **Classes**, we also consider that the instances of an **Association** can be given extensionally, or intensionally. The intension of an **Association** is given by a predicate showing which pairs of instances (from **source** and **target**) are related (i.e. which **Links** exist).

Although different kinds of **Associations** exist, we only emphasize the difference between **Aggregations** (i.e. a part-whole relationship) and those that are not **Aggregations** (i.e. those showing a class being either associated or used as attribute in another one). We consider that if **DataType** values would have identity, and could be freely associated to **Classes**, **Attributes** would just be a special case of **Association**. Thus, for the sake of simplicity, from here on, we will assume this.

There exist two ICs in the usage of **Associations**, both only concerning **Aggregations**. As depicted in Figure 3.b, **Aggregations** forming a cycle are not allowed (i.e. **Class** A being part of B, means **Class** B cannot be part of A). Moreover, between a **parent Class** and any of its **child Class** cannot exist an **Aggregation Relationship**.

Figure 3. **Aggregation** integrity constraint.

3 OPERATIONS ON OBJECT-ORIENTED SCHEMAS

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, since operations that change names do not change any form, they are not needed to conform schemas. Operations locate a particular pattern in the *source schema*, and change it to a different form in the *target schema*, leaving unaltered the rest.

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 **Relationships** (either **Association** or **Generalization**) between **Classes**.

Many occurrences of patterns appear in each schema. Conformation operations are applied successively on each pattern until a conformed schema is obtained. The conformation is the *composition of operations*, which is associative, but not commutative. We call *primitives* the atomic operations needed to *conform* schemas; the *derived* operations are those that can be obtained by composition of *primitives*. We have established a set of fifteen *primitive* operations, in the sense that none of them can be left out without affecting the set of derived operations.

Patterns of interest and *primitives* can be represented in a digraph G = (P,O), where P is the set of patterns, and O is the set of operations (Figure 4). An arc o_{ij} from p_i to p_j represents the operation that converts pattern p_i into pattern p_j. We distinguish three subgraphs corresponding to operations on G/S, A/D and inter dimensions. The reflexive arc on pattern p_6 reflects the existence of three operations acting on the same pattern, which give rise to three different forms (a, b, c), that can be seen as special cases of it.

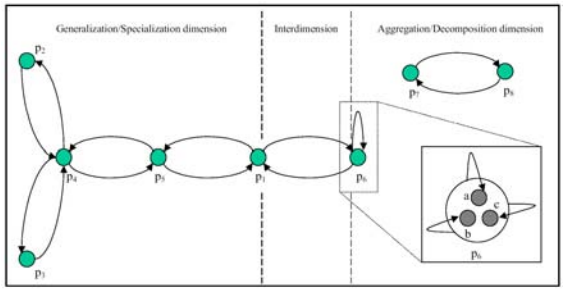

Figure 4. Patterns transition graph.

G is a symmetric graph, in the sense that for every arc o_{ij}, there is an arc o_{ji} representing the *opposite* operation, which may or may not be its mathematical *inverse*. When o_{ij} reduces the capacity of information, o_{ji} is not able to recover it, giving rise to a *target schema* different to the *source schema*; however, the patterns have the same *form*.

The matrixes depicted in Figures 5 and 10 show the *primitive* operations that work over the *patterns of interest* that we have identified for G/S and A/D

respectively, while Figure 14 shows the matrix containing *primitive* operations that combine both dimensions. The cr_i are the criteria of the **discriminators** and the n_i the names of the **Associations**. Empty cells of the matrixes represent *derived* operations.

3.1 Operations along Generalization/Specialization

There are eight *primitive* conforming operations that work exclusively along the G/S dimension (Figure 5).

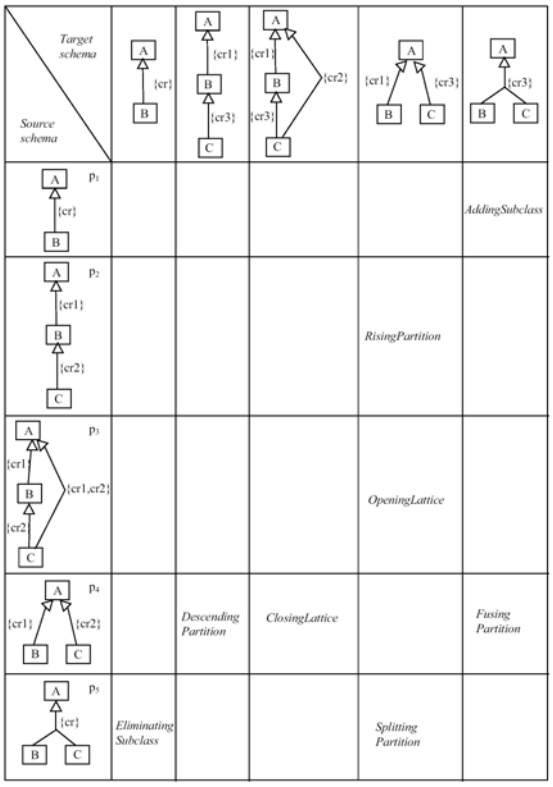

Figura 5. *Generalization/Specialization* matrix

AddingSubclass (cell c_{15}) adds a new subclass to a **Partition** by union, intersection, difference, etc. of instances of existing **Classes**, without changing the **discriminator** of the **Partition**. *EliminatingSubclass* (cell c_{51}) removes a subclass from a specialization, it can produce some loss of information capacity.

Figure 6 shows how, by difference, we add a new subclass `Others`, and it also shows how *EliminatingSubclass* removes the same subclass without producing any loss of information capacity. There are some cases where it is impossible to recover an eliminated subclass without augmenting the information capacity (for example, we cannot recover `Marine`, if we only have `Terrestrial` and `Animals`).

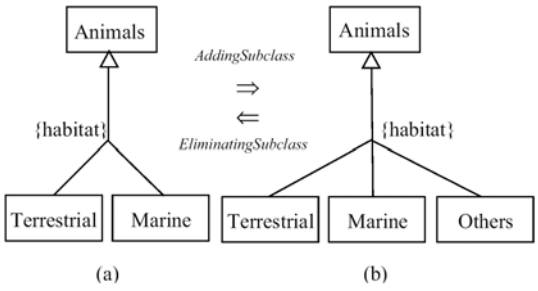

Figura 6. *Adding/EliminatingSubclass* example

RisingPartition (cell c_{24}) derives a new **Partition** of a superclass by applying the transitivity property. This operation converts an indirect subclass of a superclass in a direct one. *DescendingPartition* (cell c_{42}) does the *opposite* transformation.

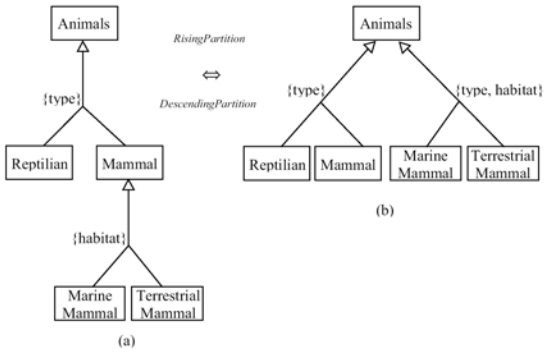

Figura 7. *Rising/DescendingPartition* example

Figure 7 shows an example of *RisingPartition*, where we obtain the subclasses `Marine Mammal` and `Terrestrial Mammal` as direct subclass of superclass `Animals`. To descend a **Partition**, its **discriminator** has to include the **discriminator** of the **Partition** of the subclass where we want to place it. So, whereas sometimes it is possible to directly apply *DescendingPartition*, in other cases it is necessary to previously use other operations to obtain an adequate **discriminator**.

OpeningLattice (cell c_{34}) changes a semi-lattice pattern into a tree pattern by removing part of one path between two **Classes**, eliminating multiple inheritance. This does not imply any loss of information capacity because, when there are two specialization paths between two **Classes**, both involve the same set of functions. The *inverse* transformation is carried out by *ClosingLattice* (cell c_{43}). Figure 8 shows an example where the **Partition**

with `habitat` **discriminator** disappears and appears.

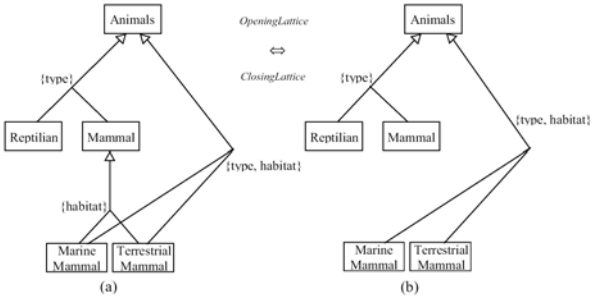

Figure 8. *Opening/ClosingLattice* example

FusingPartition (cell c_{45}) fuses all **Generalizations** in two different **Partitions** of a superclass into a unique **Partition** of it. The generated **Partition** has as **discriminator** the union of **discriminators** of the source **Partitions**. Its *inverse*, *SplittingPartition* (cell c_{54}), splits subclasses of a **Partition** into two **Partitions**. Figure 9 shows an example.

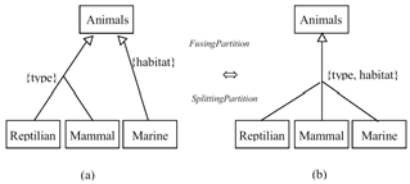

Figure 9. *Fussing/SplittingPartition* example

3.2 Operations along Aggregation/Decomposition

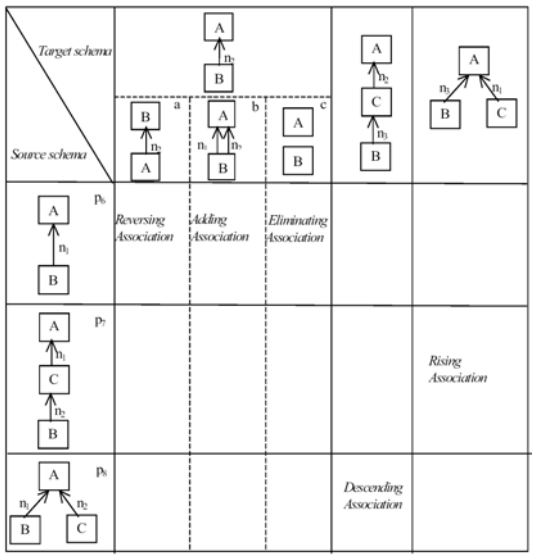

Figure 10. *Aggregation/Decomposition* matrix

The matrix associated to this dimension (Figure 10) contains three patterns (p_6, p_7, p_8). It shows conformation generic operations whose application depends on the kind of **Association**. Three operations can be applied over pattern p_6; for each one of them the form of its *target schema* is shown. The *forms* produced by *AddingAssociation* and *EliminatingAssociation* are not patterns but special cases of pattern p_6.

ReversingAssociation (cell c_{11}^{a}) changes the direction of an **Association**. Notice that it is not applicable to **Aggregations**, because it would change the meaning, not only the *form* of the schema. If it is applicable, the information capacity in the *source schema* is preserved, and itself is its *inverse*. For example, if there is an **Association** between `Cars` and `People` **Classes** showing the owner of each car (Figure 11a), the effect of this operation is to substitute this **Association** by its reversing one in the *target schema*, which shows the cars owned by each person (Figure 11b).

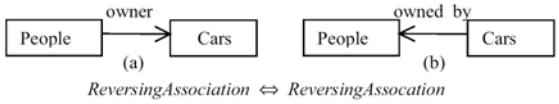

Figure 11. *ReversingAssociation* example

AddingAssociation (cell c_{11}^{b}) adds a new **Association** to the *target schema*. A new **Association** between two **Classes** can be derived by union, difference, complementarity, etc. of **Associations** between them. This operation preserves the information capacity in the *source schema*. *EliminatingAssociation* (cell c_{11}^{c}) carries out the *opposite* transformation, it eliminates an **Association** from the *source schema*; in this case, the information capacity of the *target schema* may be reduced.

In Figure 12, given a source schema with `People` and `Universities` **Classes**, and the **Associations** between them represented by means of their attributes `students`, `teaching_staff` and `adm_staff`, *AddingAssociation* derives a new **Association** in the *target schema* (Figure 12b) by union of pre-existing ones showing all people that belong to each university. The name of this new **Association** is `community`. The *target schema* in Figure 12a is obtained after applying *EliminatingAssociation* over the *source schema* in Figure 12b.

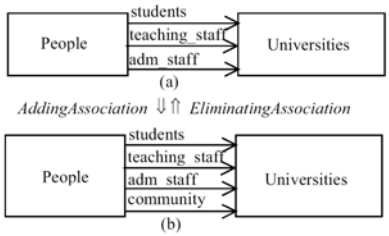

Figure 12. *Adding/EliminatingAssociation* example

RisingAssociation (cell c_{23}) derives a new **Association** between two **Classes** by applying the transitivity property. Whether the information capacity in the *source schema* is preserved in the *target schema* or not, depends on properties of **Association**s involved in the operation. The information capacity of the *target schema* may be reduced. *DescendingAssociation* (cell c_{32}) is its *opposite*; although it could seem that this operation is *derived* from successive application of *ReversingAssociation*, *RisingAssociation* and *ReversingAssociation*, this is not true, because the *ReversingAssociation* is not applicable when there is an **Aggregation**. The *target schema* obtained by *DescendingAssociation* does not always preserve the information capacity of the *source schema*.

Figure 13 depicts an example of these operations. The *source schema* (Figure 13a) includes **Classes** `Languages`, `People`, and `Clubs`; and the **Associations** among these **Classes** which show that people speak languages, and that clubs are conceived as compound objects from all their members. After performing *RisingAssociation*, a new *target schema* is obtained (Figure 13b). This schema incorporates a new **Association** connecting **Classes** `Languages` and `Clubs`. Therefore, in the *target schema*, we can obtain for a given club all its members and all the languages spoken in the club, but data about which languages were spoken for each person has been lost. The *target schema* in Figure 13a is obtained after applying *DescendingAssociation* over the *source schema* in Figure 13b; the new **Association** represents, for each person, the languages that are spoken in the clubs to which the person belongs instead of the languages spoken by the person.

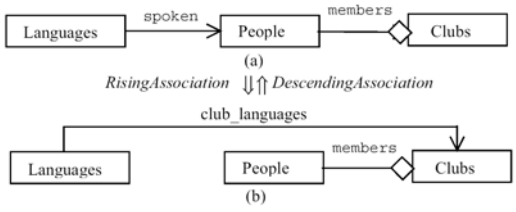

Figura 13. *Rising/DescendingAssociation* example

3.3 Interdimension Operations

Figure 14 shows the *primitive* conforming operations that work simultaneously along the G/S dimension as well as the A/D dimension.

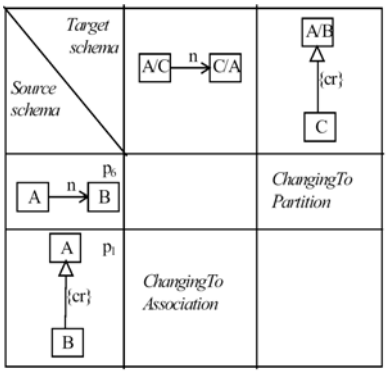

Figure 14. Interdimension matrix

ChangingToPartition (cell c_{12}) transforms an **Association** into a **Partition**, so one or more subclasses are added to the *target schema*. The contents and number of these subclasses depend, respectively, on the set of objects of the **Class** that is being specialized and the **Association** that is being transformed in a **Partition**. The **Class** to be specialized could be any of the two **Classes** related by the **Association**. Given that no new information can be added, the subclasses only incorporate those **Associations** that are inherited from their superclasses; the *target schema* preserves the information capacity of the source schema. *ChangingToAssociation* (cell c_{21}) carries out the *opposite* transformation, it converts a **Partition** of the *source schema* into an **Association** (whose direction can be chosen at will) in the *target schema*. Moreover, the *target schema* disregards all the subclasses that participated in the **Partition**. In general, the *target schema* has less information capacity.

Figure 15 shows an example of application of these operations. Given a *source schema* (Figure 15a) with **Classes** `Sex` and `People`, the **Association** between them represents the sex of each person. Assuming that there are people having associated object 'female' and people having associated object 'male', the subclasses `Women` and `Men` will be added to the *target schema* (Figure 15b) using *ChangingToPartition* . The set of objects of these subclasses are, respectively, those people whose sex is 'female' and whose sex is 'male' in the *source schema*. Finally, it is important to note that **Class** `Sex` does not disappear from the schema

because it could be **Associated** to other **Classes**. Figure 15a is obtained after applying *ChangingToAssociation* over the *source schema* in Figure 15b.

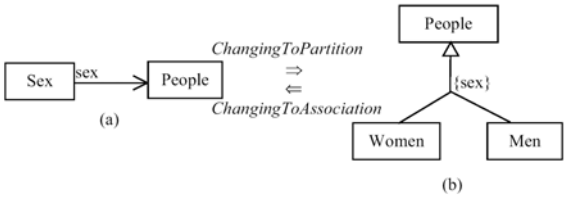

Figure 15. *ChanginToPartition/Association* example

4 RELATED WORK

Schema conformation 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 modelled Universe of Discourse; hence, these changes can augment the information capacity. 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.

4.1 Operations on Object-Oriented Data Models

Directly related to the Schema Conformation problem is the proposal found in Motro (1987), where a set of operations to build and modify the generalization hierarchy is presented. In Mannino *et al.* (1988) generalization hierarchies can be merged using a set of operations.

Several view mechanisms have been proposed for O-O models, as can be seen in Motsching-Pitrik (1996). In Rundensteiner (1992) proposal, Derived Classes are defined using Object Algebra, and then integrated in an O-O schema (the Global Schema) to define External Schemas as subschemas of it. In Bertino (1992), Derived Classes that include non-derived attributes can be defined using a specific language. In Santos *et al.* (1994) an External Schema definition methodology is proposed; in it, Derived Classes can contain existing or new objects. In Rundensteiner *et al.* (1998) a Schema Evolution mechanism based on a View Definition system is proposed, and it has no limitation on the accepted changes. In proposals where External Schemas are built from the Conceptual or other External Schemas without augmenting their information capacity, the conformation operations presented in this paper can be directly used. As mentioned in Miller *et al.* (1993), one of the operational objectives of defining External Schemas is to restructure information to be integrated in others schemas, which is also the target of our work.

In Banerjee *et al.* (1987), a taxonomy of primitive operations for Schema Evolution in the Orion system is defined; each operation has a semantic based on a set of rules that preserve Schema Invariants. For the GemStone system (Penney and Stein, 1987), a set of primitive operations, based on Schema Invariants as well, is defined; in this case, the object model is simpler than the Orion one (since multiple inheritance is not allowed), and this fact is reflected on operations. The corresponding proposal for the O_2 system can be found in Zicari (1992). In Claypool *et al.* (1998) a framework is proposed based on the Object Data Management Group (ODMG) data model; in it, complex schema evolution operations can be defined as a sequence of primitive ones. In these models, subclass relationships are defined based exclusively on subtype relationships, without taking into account any specialization criteria; for this reason, the primitive operations defined along the G/S dimension perform only the addition or removal of subclass relationships. Related to the A/D dimension, the only association in these models corresponds to the definition of a class as an attribute domain; therefore, the only operation along this dimension is devoted to change attribute domains. In relation to the G/S and A/D dimensions, in this paper we have defined a set of operations wider than the above mentioned, the reason is that we consider specialization criteria and associations.

4.2 Operations on other Data Models

Operations to transform or restructure schemas have also been proposed in others data models. The complex object model put forward in Abiteboul and Hull (1988) allows the definition of typed hierarchical objects; in this environment, a set of operations over this kind of objects, based on rewriting rules, is defined. These operations work along the A/D dimension, since they restructure the type of a complex object by associating its components; most of them preserve the information capacity, only two augment it.

In Kwan and Fong (1999), a schema integration methodology is put forward, it is based on the Extended Entity-Relationship model, but it could also be applied to the Relational model. This methodology offers a set of rules to solve semantics conflicts and to merge entities and relationships taken from different schemas in another schema. These rules operate along the G/S or A/D dimensions; the generated schema always preserves the information capacity.

5 CONCLUSIONS

Conforming operations change a pattern in a *source schema* into a different pattern in a *target schema*, thus changing the *form* of the schema, without augmenting the information capacity. We have presented a set of *primitive* conforming operations on Object-Oriented database schemas; these operations may be represented by arcs of a graph and by cells of a "pattern×pattern" matrix. Other conforming operations can be *derived* from these *primitives*.

The use of different O-O dimensions allows us to classify the operations into three groups, each one with its matrix: operations along the *Generalization/Specialization* dimension, operations along the *Aggregation/Decomposition* dimension, and operations changing from one dimension to the other. Each operation produces a *mapping* between its *source* and its *target schema*. These *mappings* lie along the *Derivability* dimension, and is subject of our research in progress, that will give rise to the implementation of a case tool for schema integration.

Acknowledgments

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

REFERENCES

Abiteboul, S., Hull, R., 1988. Restructuring Hierarchical Database Objects. *Theoretical Computer Science 62*. North-Holland.

Banerjee, J. *et al.*, 1987. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In *ACM SIGMOD'87*.

Bertino, E., 1992. A View Mechanism for Object-Oriented Databases. In *EDBT'92*, *LNCS 580*. Springer.

Castellanos, M. *et al.*, 1994. Semantically Enriching Relational Databases into an Object Oriented Semantic Model. In *DEXA'94*, *LNCS 856*. Springer.

Claypool, K. *et al.*, 1998. SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework. In *CIKM'98*. ACM Press.

Conrad, S. *et al.*, 1999. Engineering Federated Information Systems. *ACM SIGMOD Record*, *28*.

García-Solaco, M. *et al.*, 1996. Semantic Heterogeneity in Multidatabase Systems. In *Object Oriented Multidatabase Systems*. Bukhres, O., Elmagarmid, A. (eds.), Prentice-Hall.

Kwan, I., Fong, J., 1999. Schema Integration Methodology and its Verification by Use of Information Capacity. *Information Systems*, *24*. North-Holland.

Mannino, M. *et al.*, 1988. A Rule-Based Approach for Merging Generalization Hierarchies. *Information Systems*, *13*. North-Holland.

Miller, R. *et al.*, 1993. The Use of Information Capacity in Schema Integration and Translation. In *VLDB'93*. Morgan Kaufmann.

Motro, A., 1987. Superviews: Virtual Integration of Multiple Databases. *IEEE TSE 13*. IEEE Press.

Motsching-Pitrik, R., 1996. Requirements and Comparison of View Mechanisms for Object-Oriented Databases. *Information Systems 21*. North-Holland.

OMG, 2002. *Unified* Modelling *Language Specification*. Version 2.0. http://www.omg.org/uml/

Penney, D., Stein, J., 1987. Class Modification in the GemStone Object-Oriented DBMS. *ACM SIGPLAN Notices 22*. ACM Press.

Rundensteiner, E., 1992. MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases. In *VLDB'92*. Morgan Kaufmann.

Rundensteiner, E. *et al.*, 1998. Capacity-Augmenting Schema Changes on Object-Oriented Databases. In *OOIS'98*. Springer.

Saltor, F. *et al.*, 1991. Suitability of Data Models as Canonical Models for Federated DBs. *ACM SIGMOD Record 20*.

Santos, C. *et al.*, 1994. Virtual Schemas and Bases. In *EDBT'94*, *LNCS 779*. Springer.

Sheth, A., Larson, J., 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys 22*.

Zicari, R., 1992. A Framework for Schema Updates in an Object-Oriented Database System. In *Building an Object-Oriented Database System*. Bancilhon, F., Delobel, C., Kanellakis, P. (eds.), Morgan Kaufmann.