A Canonical Model for the Interoperability among Object-Oriented and Relational Databases

Malú Castellanos, Fèlix Saltor and Manuel García-Solaco
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, Barcelona
{castellanos,saltor,mgarcia}@lsi.upc.es

Abstract

In the interoperability context, autonomous pre-existing databases are federated at some degree in order to share their data. The growing importance that the interoperability among object- oriented and relational databases is gaining, makes essential the development of adequate data models that serve as canonical models for the federation. The canonical model constitutes the basic building block for methodologies and tools for the detection of interdatabase semantic relationships that is required to access the federation in an integrated way. This paper presents one such canonical model and points out its usage for coupling relational and object-oriented databases thru a semantic enrichment process.

1 INTRODUCTION-MOTIVATION

Relational and object-oriented (O-O) approaches are the two paradigms that have gained wider acceptance in the database field during the last years. Databases based on these paradigms will proliferate and the need to access them in an integrated way will arise as a natural consequence. Several architectures have been proposed for this purpose [Sheth, 1990], where the databases are tightly or loosely coupled in a federation. In any of them, the semantic relationships that exist among the databases must be identified for the integrated access to the federation. This is a critical task complicated by the semantic heterogeneities that exist among these databases and the lack of an exact understanding of the meaning of their contents. As a first step to solve this problem, data models and methodologies based on these models must be developed. This is the aim of the BLOOM project[1] whose data

[1] Research partially supported by the spanish PRONTIC program, project TIC89/0303

model is described in section 2. The methodology includes a semantic enrichment step for upgrading the semantic level of the local schemas in order to facilitate the detection of the interdatabase relationships, as well as to couple relational and object-oriented databases as discussed in section 3. The final result of the project will be a tool that semi-automatizes the overall process and can serve as a testbed for experimenting with it.

2 AN OVERVIEW OF THE BLOOM MODEL

2.1 CONSTITUENTS OF THE MODEL

BLOOM [Castellanos, 1991b] is a semantic extension of an object-oriented model which complies with the essential features of suitability of canonical data models for interoperable databases reported in [Saltor, 1991]. The main constituents of the model are objects, types, classes and semantic abstractions.

Objects: they represent real world objects and are instances of ADTs manipulated by means of their interface. Object identity is mantained by the system. We distinguish primitive objects, considered elementary and provided by the system, from derived objects whose type is user defined deriving it from other types.

Types: describe the structure and behaviour of their object instances, according to the ADT concepts. Types are arranged in a type semi-lattice which reflects the subtype relationship "is_a". Inclusion semantics is used for subtyping and multiple inheritance is supported. An object may be an instance of more than one type where each one of them provides a certain view on the object.

Classes: a class is a set of objects (members) associated with a type. There may be more than one class of a type. An object may be member of multiple classes and/or instance of multiple types. A subclass relationship between two classes is a combination of the subtype relationship among their types and the subset relationship among their extents.

Semantic Abstractions: the description of the structure entails the description of the relationships which are characterized by the abstractions that they represent:

- Classification/Instantiation
- Four kinds of Specialization: complementary, disjoint, alternative and general
- Three kinds of Aggregation: simple, collection and association

The operational functionality models the dynamic properties which can range from the specification of insertion and deletion constraints to the modeling of operations. These constraints used to maintain the integrity of the semantic database, are embodied in the metaclasses that specify the behavioural interpretation of the semantics of the model. There are inherent constraints for each abstraction (section 2.3 and 2.4), as well as inherent constraints for different kinds of specific existence dependencies (section 2.2).

Metaclasses are a useful mechanism for adapting the model to different contexts by extending it with new metaclasses and for specifying not only the behaviour of their class instances, but also that of the instances of their instances as in [Klas, 1990].

2.2 EXISTENCE DEPENDENCIES

a) **Strict Existence Dependencies**: as its name suggests, the existence of the dependent object strictly depends on the existence of the one on which it depends,

called dependor here. This dependencies can reflect a situation where an object cannot exist in the real world independently of another one, or else an interest dependency where the dependent continues existing but it is not of interest anymore.

The insertion constraint linked to this concept makes sure that the dependent object cannot be created if the object upon which it depends has not been created before. With respect to the deletion constraint, this can have two different implications: in one case the deletion of the dependor object entails the deletion of the dependent object, this corresponds to the *Propagate_Strict_Dependency* case, and in the other case called *Block_Strict_Dependency* the deletion of the dependor simply cannot be made if there are objects dependent on it. Each one of these dependencies can be classified as *Exclusive, Shared* or *All* according to the number of instances of the dependor object type on which an object depends.

b) **Relaxed Existence Dependencies**: as its name suggests, it is a relaxed version of the strict dependency. The idea is that the dependent object cannot be created if the object on which it depends doesn't exist, but once it is created, its existence turns out to be independent of it. Therefore, its implications are only with respect to insertion, since the deletion of the dependor has no impact on the dependent. *Exclusive_, Shared_* and *All_Relaxed_Dependency* are distinguished here.

2.3 THE GENERALIZATION/SPECIALIZATION DIMENSION

All semantic models capture the notion of generalization/specialization but in BLOOM we go one step further by distinguishing different types of specialization in order to express yet more semantics: *Disjoint_, Complementary_, Alternative_* and *General_Specialization.*

As in any other O-O model, two kinds of insertion are distinguished: "new" and "add". Their semantics are specified in the insert constraints of the metaclass corresponding to the Generalization/Specialization abstraction. The "new" operation on a subclass implies an insert on the superclass as well. In contrast, the "add" operation on a subclass implies an insert on the superclass only if it didn't exist there previously.

For deletion, we also distinguish two kinds, one called "kill" that makes the object disappear definitively from the object base, propagating its deletion to its super and subclasses. The other one called "remove", only removes the object from the class indicated and from its subclasses.

According to the type of specialization, these constraints are further refined as follows:

a) **Disjoint Specialization**: the specialization subclasses are disjoint, that is, each object of the superclass belongs at most to one subclass. The only implication that this restriction has is on the "add" operation where we must make sure that the object being inserted doesn't exist in another sibling subclass.

b) **Complementary Specialization**: in this case the subclasses are complementary and overlapping, that is, each object of the superclass belongs at least to one subclass. Then, when we make "add" or "new" in the superclass, this insertion must be propagated to at least one subclass (system prompts for it). In contrast when there is a delete in a subclass and the object belongs to only one of the complementary subclasses, then either the deletion is blocked or else it is propagated to the superclass. This leads to two kinds of complementary specialization: the *Blocks_* and the *Propagates_Complementary_Specialization.*

c) **Alternative Specialization**: in this case, the subclasses are complementary and disjoint, that is, each object belongs to one and only one subclass. The "add" or "new" operations on a superclass has the same effect as in the complementary specialization, but when the insertion is on a subclass, we must check that the object doesn't exist in a sibling subclass. The deletion of a subclass can be blocked or propagated giving rise to the *Blocks_* and *Propagates_Alternative_Specialization*.

d) **General Specialization**: has no restrictions and no further implications.

2.4 THE AGGREGATION DIMENSION

Aggregation [Smith, 1977] in general, is the abstraction by which several objects are aggregated into a new, complex object. In BLOOM we distinguish three types of aggregation according to the different semantics found in the real world, one simple and two complex: *Simple_, Collection_* and *Associaton_Aggregation*

a) **Simple Aggregation**: objects $O_i, \ldots, O_n$ are *simple_aggregated* to another object O_j, just to model a property of it. In the real world, O_j is not the result of the aggregation of $O_i, \ldots, O_n$, which constitute merely its properties of interest in the UoD. In this sense, aggregation provides a means for specifying the attributes of an object type. This is called "reference" by some authors and 'weak reference' in [Kim, 1989].

Attributes can be primitive or derived depending on their types, single-valued (by default) or multi-valued (*set_of*), *obligatory* or not (by default). A dependency may exist between an object and another one *simple_aggregated* to it. The kind of this dependency can be any one of those explained before and must be specified thru the corresponding construct.

b) **Collection Aggregation**: Contrasting with the previous one, an object type O' *collection_aggregated* to another type O, is not only a simple multi-valued property of O, but it is precisely this property what gives rise to O. This abstraction permits to treat a collection of objects as another object. This concept, more commonly known by cover aggregation, was introduced by Hammer and McLeod who simply called it "aggregate". In turn, the *simple_aggregation* concept is used to specify the properties of the collection.

The behaviour of this abstraction is given by the *shared_strict_dependency* because the existence of the aggregate object strictly depends on the existence of the collection of objects that constitute it. Two kinds of collection aggregation are distinguished: *blocks_collection* and *propagate_collection*, depending on whether the deletion of the last member of the collection is blocked or entails the deletion of the aggregate object.

c) **Association Aggregation**: the aggregate object is formed by associating one (or more) object instances of the types involved in the association. The associated objects are not simply properties of the aggregate one, but it is their association what gives rise to it. The aggregate object, in turn, has its own properties specified by the *simple_aggregation* abstraction explained before.

The behaviour of this abstraction is given by the composition of the different kinds of *strict_dependency* linked to the relationship between each component and the aggregate object. Two different situations can arise. One where the aggregate object cannot exist if any of its components objects doesn't exist. The other situation is similar but here it is possible to delete a component object as long as it is not the last object of one of the associated types.

2.5 OTHER FEATURES OF BLOOM

Next we comment some characteristics related to the semantic relativism of our model. BLOOM provides a set of integration operators including discriminated operators [García-Solaco, 1991], [Saltor, 1992] like the discriminated generalization, to support multiple semantics in a federated schema, and to overcome schematic discrepancies by transforming metadata into data and vice versa. Besides downward inheritance, BLOOM also includes upward inheritance as in [Schrefl, 1988], by which global types obtained from the integration of local ones, inherit the structure and behaviour of these last ones. This facilitate the task of definition of federated schemas.

3 COUPLING OF RELATIONAL AND O-O DATABASES

The BLOOM model has been conceived in the context of database interoperability and it easies the task of coupling relational and object-oriented databases. In [Castellanos, 1991] we have described how the semantic level of relational schemas is upgraded thru a semantic enrichment process based on inclusion dependencies in their whole generality. There, we can contrast the poor expressiveness of the relational model against the richness of BLOOM.

Since BLOOM is an O-O model, the conversion from database schemas expressed in other O-O models to the richer set of abstractions of BLOOM results less complicated. Object-oriented models incorporate much more semantics than the relational one; at least every O-O model includes specialization and the set and tuple constructors. Furthermore, semantic integrity constraints are encapsulated inside the definition of the object type and they can be directly analyzed to obtain the necessary to identify the corresponding BLOOM abstractions.

Once object-oriented and relational schemas have been converted to rich BLOOM schemas, they can be effectively coupled by our schema integration methodology. Integration of database schemas involves a comparison task that searches for the semantic relationships that exist among them. The complexity resulting from the excessive number of comparisons that have to be performed, is due to the fact that the search is not guided at all. Each object in one schema must be compared with every object in the other schema, and these comparisons are in turn decomposed in comparisons between their attributes. In contrast, in our approach [García-Solaco, 1992] the complexity is reduced because the search is guided by the structures of the generalization and aggregation semi-lattices that conform the BLOOM schemas. Many non meaningful comparisons are eliminated and the most promising ones are identified. A systematic procedure for analyzing the structures in a meaningful way is given. The strength of the technique stems from the formal and precise semantics of BLOOM.

4 CONCLUSIONS

A model captures the semantics of the UoD by means of the abstractions it supports in the generalization/specialization and aggregation dimensions, and how these abstractions are structured. Because BLOOM is based on a rich set of semantic concepts, which distinguish different kinds of dependencies, specializations and aggregations, it can capture more semantics than most existing models. This allows the definition of object bases with type structures which are really expressive.

As stated in section 3, in the context of interoperability we need a canonical model

rich enough to capture the semantics already expressed in the local schemas and the one captured by the enrichment process. Furthermore, we need an adequate model by means of which O-O and relational databases can be coupled. BLOOM has been designed for this purpose. Its practical significance in the semantic enrichment, detection of interdatabase semantic relationships and conformation phases of the integration process make of BLOOM the key component of our approach.

References

M.Castellanos and F.Saltor. (1991) "Semantic Enrichment of Database Schemas: an Object-Oriented Approach". In *Proceedings 1st. Int. Workshop on Interoperability in Multidatabase Systems*, Kyoto.

M.Castellanos, F.Saltor and M.García-Solaco. (1991b) "The Development of Semantic Concepts in BLOOM Using an Object-Oriented Metamodel". Report LSI-91-22. U.P.C., Barcelona.

M.García-Solaco, M.Castellanos and F.Saltor. (1992) "Discovering Interdatabase Resemblance of Classes for Interoperable Databases". Submitted to RIDE-IMS93.

M.García-Solaco and F.Saltor. (1991) "Discriminated Operations for Interoperable Databases". In *Proceedings 1st. Int. Workshop on Interoperability in Multidatabase Systems*, Kyoto.

W.Kim, E.Bertino and J.Garza. (1989) "Composite Objects Revisited". In *Proceedings ACM SIGMOD Conference on the Management of Data.*

W. Klas, E. Neuhold and M. Schrefl. (1990) "Metaclasses in VODAK and their Application in Database Integration". In *Arbeitspapiere der GMD No.462*, GMD.

F.Saltor, M.Castellanos and M.García-Solaco. (1991) "Suitability of Data Models as Canonical Models for Federated Databases". *ACM SIGMOD Record*, 20(4).

F.Saltor, M.Castellanos and M.García-Solaco. (1992) "Overcoming Schematic Discrepancies in Interoperable Databases". In *Proceedings of the DS-5 Semantics of Interoperable Database Systems.*

M.Schrefl and E.Neuhold. (1988) "Object class definition by generalization using upward inheritance". In *Proceedings of the 4th. Int. Conference on Data Engineering*, Los Angeles. IEEE-CS Press.

A.Sheth and J.Larson. (1990) "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases". *ACM Computing Surveys*, 22(3).

J.Smith and D.Smith. (1977) "Database Abstractions: Aggregation and Generalization". *ACM TODS* 2(2).