In: G.E.Lasker (ed.): Advances in Cognitive Engineering and Knowledge-Based Systems,
Proc. of the Intl. Symposium on Systems Research, Informatics and Cybernetics, Baden-Baden,1993

View Concepts for Object-Oriented Databases

Michael Dobrovnik and Johann Eder

Institut für Informatik
Universität Klagenfurt
Universitätstr. 65
A-9020 Klagenfurt / Austria
email: {michi, eder}@ifi.uni-klu.ac.at

Abstract

We present a concept to introduce external models in object-oriented databases to regain the traditional three level architecture of database systems consisting of an internal, a conceptual and several external models. In contrast to other approaches our concept takes into account all traditional features of external models - such as submodeling, interfacing application programs and databases, logical data independence, canned queries, and individualized access and security management. A unique advantage of this concept is the clear separation of the type and class hierarchies of the external models from those of the conceptual model allowing better and cleaner modularization of information systems built on top of object-oriented databases.

1. Introduction

The concept of external models in database systems is used for various purposes. First of all it represents the view of a user or an application program on the database, i.e. on the conceptual schema. In this role an external model is a submodel of the conceptual model in the sense that the external model can be deduced from the conceptual model. So the definition of a view is a mapping of the terms and concepts of the conceptual model to those of the respective user. External models are also the interface specification between database systems and application programs providing logical data independence and thus reducing the maintainance effort in case of changes to the conceptual schema. In particular, the introduction of this additional layer permits the flexible and modular architecture of application systems built upon a database. Furthermore, views are the basis for specifying, managing and enforcing access control by stating which user may apply which operations on which data. In most database systems the notion of views stands also for derived database objects, where the view is a named query expression, facilitating the formulation of queries. So we can conclude, that external models are an important feature of database systems.

The introduction of the concept of external models in object oriented database systems has long been considered as a major problem, in particular, because of the lack of declarative query languages, because views may confuse the type or class hierarchy, and views seemingly did not integrate well with the object-oriented concepts developed for programming languages. Nevertheless, the concept of views is important to build large application systems on top of an OODBMS.

In embedded systems were an OODBMS is used for a system serving a single, relatively narrow task, the problems arising from the intermix between application and objects may be negligible. But consider the OODBMS to be the main building block of an enterprise wide information system consisting of a large number of relatively independent application systems, each one of these systems having different and diverse needs. The spectrum of the dynamic behavioural requirements is much wider and richer than merely the variation in the object structures (which nevertheless may be complex too).

Recently, several approaches to integrate views in OODBMS have been reported (e.g. [AB91, Bert92, HZ90, SLT91]). However, none of these addresses all the features outlined above. In the paper we propose a general concept for the introduction of views in OODBMS which serves all of the above mentioned purposes.

The rest of the paper is organized as follows: In section 2 we give an overview of the data model we use for presenting our ideas. In particular we emphasize the distinction between types and classes. In section 3 a general concept for the introduction of views in OODBMS is introduced. Finally, in section 4 we discuss this approach and draw some conclusions. In general we assume that the reader is familiar with the basic concepts of object oriented databases as described for example in [ABC+89].

2. A Sketch of the Underlying Data Model.

In this section we briefly describe the data model on which our considerations are based to the extent necessary for introducing the view concept. This data model is associated with a schema definition language and a query language. Both are beyond the scope of this paper. A schema in our data model consists of type definition and class definitions.

Like in other approaches we distinguish between types and classes. Types represent intensional information, while classes denote extensional information. So types describe the structure of objects and values, in particular, which components (instance variables) they consist of, and which methods can be applied to them, and the signature as well as the implementation of methods.

Types are defined in the context of an inheritance lattice for structural inheritance. If we define a type t to be subtype of a type s, we mean, that the type t inherits all the structural information (components, signature and implementation of methods) from its supertype. In the definition of the subtype we can add additional components and additional methods. Components and methods inherited from the supertype can be overridden whereby we assume that the inheritance lattice forms a subtype hierarchy in the sense of covariant subtyping, i.e. wherever we can use an object of a certain type we also can use an object of one of its subtypes.

The class concept is closely tied to oo-technology since the very beginning [GR83]. However, types and classes usually have not been distinguished resulting in a mix of structural and extensional information. Here we see classes from a different angle than the more traditional approaches.

We define classes to be object containers, which can include objects compatible with a ground type. The instance of a class is the object set of the class. There may be no class, exactly one class or many classes of a certain type. So, there is no class automatically generated when a type is defined. Operators on classes are provided to include an object in a class, to remove it from the class, to test if an object is in a class and to iterate over the object set of the class.

Classes can be arranged in a class hierarchy. Subclassing means subsetting the object set of a class. An object in a certain class C is also in all superclasses of C (instance inheritance). But additionally, an object may be in any number of unrelated classes if the ground type of the class is compatible with the object's type. The membership of an object to a class can be made explicit by means of the operators mentioned above. The second possibility to define the object set of a class is a predicate associated with the class. Predicative subclassing is a powerful means of automatically (re-)classifying objects.

The key concept of persistence is attached to classes in our model. An object is persistent, if it is a member in (at least) one class or if it is referenced by a persistent object. Persistence thus is defined by reachability from certain persistence roots (the classes). If the last reference to an object is removed, the object is deleted from the database (garbage collection semantics).

The separation of types and classes is crucial for the cleanliness of the type and class hierarchies when we introduce views.

The schema definition language furthermore consists of a language for the implementation of the methods and of query language. The method implementation language is a Turing complete procedural programming language which also can contain expressions of the query language like in O2 [BDK92]. Similar to [LS92] the query language offers generic operations for projection, selection, extension, join, and set operations. Without going into details we briefly present the semantics of these operations. Selection returns a set of objects which validate a selection predicate. Projection returns a set of objects but drops components of these objects. Extension returns a set of objects adding components derived by methods or other (associated) objects. Join returns a set of objects formed by combining the components of other objects.

Query expressions may be object preserving, i.e. the objects of the result set have the same object identifiers as the original objects. They may be object generating, where the objects in the result set have new object identifiers, or value generating, i.e. the result set does not consist of objects but of (maybe complex) values.

3. Views for OODBMS

External schemas are introduced to provide a layer between applications and users on one side and the conceptual level on the other side [TK78]. The external schema definition explicitly states which parts of the conceptual schema are needed by the application and in which form they are needed. External models also serve as a logical buffer to minimize impacts of schema evolution, be it the conceptual or the external schema.

An external schema is based on a conceptual schema and consists of views, which are derived classes and types. The derived classes are constructed from elements in the conceptual schema (types, classes) and from components defined in the external schema themselves. Types of the external schema can be refined and used to build classes upon them. There are certain restrictions on possible specializations, but it is certainly possible to add new methods and to override existing methods.

The views are derivations of classes in the conceptual schema based on a query which can be seen as a specification of how to compute the extent of a view (i.e. the collection of objects it consists of).

The hardest problem reported for views (seen as named query expressions) is the insertion of the derived type of the result of the query in the type hierarchy [SLT91]. For example, if we perform a projection on a collection of objects we omit attributes of a type. So the type of the resulting set of objects has fewer attributes than the original one. Thus, it should be a supertype of the type of the original class. If we would insert this derived type in the type lattice, we face the following problems: First, we have to identify the position where this type has to be inserted - which is far from trivial. Second, if in the original type some methods have been overridden, we end up with the cumbersome upward inheritance, where a supertype (the derived type) inherits methods and attributes from a subtype (the original type). A different solution would be to insert the derived type as a restricted subtype of the original type. In this case, the subtyping property of the lattice gets lost, i.e. it is no longer possible to use an object of subtype wherever an object of a supertype is expected.

In our approach we avoid this problem throughout by introducing a different type lattice for each external model. The definition of an external model now is the specification how the type lattice of the external model is derived from the conceptual model. So, like in the relational model, we differentiate between an external model, which is a representation of the universe of discourse form a user specific viewpoint, and the mapping of this model from the conceptual model.

The definition of an external model consists of a type lattice, a class lattice and the mappings from the types and classes of the conceptual model to those of the external model. User or application programmers only see the two lattices but not the mappings. Thus, they see a coherent type and class hierarchy without the problems sketched above. And the user can apply queries against this external model. For application programs this external model is the interface definition to the database, and the type and class hierarchy will be part of the type and class hierarchy of the application program.

Types in the external model are specified with the same schema definition language used to formulate the conceptual model. The definition of an external type can refer to components of the conceptual model. In the definition of external types, the name of the external type and its external supertypes are given. Furthermore, the type can be constructed with the usual means (components and methods), or it can reference a type in the conceptual model. In the referencing process, it is possible to project away certain aspects (components and/or methods) of the conceptual type, or to define additional components and/or methods, to build an extension of the type in the conceptual model. To some extent, this referencing mechanism to conceptual types resembles the inheritance concept in oo-type systems. There we do not have to write down all components and methods of a type but reference a supertype and inherit all of its components and methods. However, the reference mechanism for the definition of external models is more powerful, as it allows greater flexibility in referencing parts of the type hierarchy of the conceptual model. The two mechanisms serve two rather different purposes. While structural inheritance mainly increases reuseability, the interlevel referencing between the schemas is a means to achieve the layered model architecture. Here, the external model uses parts of the conceptual level in a way best suited for the external point of view, without interfering the two layers. The whole architecture gains transparency, gets easier to use and is better to understand, and therefore also easier to maintain.

The types in the external model are used to define the classes at the external level. We specify the name of the class, its external ground type, its external superclasses, and associate it with a query expression which references classes of the conceptual model. Those external classes now are the views specifically designed and carefully tailored for the requirements of a certain type of users and applications of the database. The views serve as entry points for queries and operations on the database and are the single way to access the objects in them.

An important aspect of this approach is the consistency of an external model. In particular, it is necessary to check, whether the subtype relationships stated in the definition of the external model are valid when the references to the types of the conceptual model are incorporated. In particular naming and subtyping conflicts between external inherited and referenced names and types have to be resolved. A consistency checker is an obligatory part of the definition facility for external models.

4. Conclusion

We presented a view concept for object oriented database systems, which serves as an interface between application programs and the conceptual model implemented in the OODBMS. This additional level of abstraction introduces a new dimension of modularity of information systems built on top of object oriented databases. The conceptual enterprise-wide level and the narrower application level can be better separated and there is no need to intermix conceptual and application specific information in one single object (class).

The conceptual model is decoupled from the applications and vice versa. The external model specifies which part of the conceptual model is used by an application or an application class, thereby documenting dependencies. Furthermore, (some) implications of schema changes for application systems can easily be deduced from the external model specification.

A crucial point is the smooth integration of the external and the conceptual model. Access to the database must be fully transparent in the application (as far as the external model permits).

In this paper we could only sketch the basic ideas of this view concept. More advanced and detailed topics like view updates, method invocation, etc. had to be skipped. The exact definition of the schema definition language together with a formalization of the semantics of the referencing mechanism, and the consistency of external models is subject of a forthcoming paper.

References

[AB91] S. Abiteboul, A. Bonner: Objects and Views. In: Proc. ACM SIGMOD, Denver, 1991, pages 238-247.

[ABC+90] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik: The object-oriented database systems Manifesto. In: W.Kim, J.-M. Nicholas, S. Nishio (eds.): Proc. 1st Intl. Conf. on Deductive and Object-Oriented Databases - DOOD '89, Elsevier, Kyoto, 1989, pages 40-57

[BDK92] F. Bancilhon, C.Delobel, P. Kanellakis (eds.): Building an Object-Oriented Database System - The Story of 0_2.Morgan-Kaufmann, San Mateo, 1992.

[Bert92] E. Bertino: A View Mechanism for Object-Oriented Databases. In: A. Pirotte, C. Delobel, G. Gottlob (eds.): Advances in Database Technology, - EDBT'92, Springer Verlag, 1992, pages 136-151.

[GR83] A. Goldberg, D. Robson: SMALLTALK-80 - The Language and its Implementation. Addison-Wesley, Reading, 1983.

[HZ90] S. Heiler, S. Zdonik: Object Views: Extending the Vision. In: Proc. IEEE Int. Conf. on Database Engineering, Los Angeles, 1990, pages 86-93.

[LS92] C. Laasch, M. Scholl: Generic Update Operations Keeping Object-Oriented Databases Consistent. In: R. Studer (ed.): Proc. 2nd GI-Workshop on Information Systems and Artificial Intelligence (IS/KI), Springer-Verlag, Ulm, 1992.

[SLT91] M. Scholl, C. Laasch, M. Tresch: Updateable Views in Object-Oriented Databases. In: C. Delobel, M. Kifer, Y. Masunaga (eds.): Proc. 2nd Intl. Conf. on Deductive and Object Oriented Databases- DOOD '91, Springer-Verlag, Munich, 1991, pages 190-207.

[TK78] D. Tsichritzis, A. Klug (eds.): "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Database Management Systems. Information Systems 3 (1978).