in: L.Gün, R.Onvural, E.Gelenbe (eds.): Proc. 8th International Symposium on Computer and Information Systems, Istanbul, 1993

A Concept of Type Derivation for Object-Oriented Database Systems

Michael Dobrovnik and Johann Eder

Institut für Informatik,
Universität Klagenfurt
Universitätsstr. 65
A-9020 Klagenfurt / Austria

email:{michi,eder}@ifi.uni-klu.ac.at

Abstract - We present a concept of type derivation in order to introduce external models in object oriented database systems. This leads to the traditional three level DBMS architecture, 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. We provide a clear separation of the type and class hierarchies of the external schema from those of the conceptual schema. This approach allows for better and cleaner modularization of information systems built on top of object-oriented databases. In this paper, we concentrate on the mechanisms to separate and connect the external and conceptual type hierarchies.

1. INTRODUCTION

External models in database systems are used for a number for varying purposes. An external model represents the view of a user or an application program on the database, i.e. on the conceptual schema. The external model is a submodel of the conceptual model and can be deduced from it. A view is a mapping from 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 maintenance 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 introduced by 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 behavioral 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. [1,4,7,9]). However, none of these addresses all the features outlined above. We propose a concept for the separation and integration of external and conceptual schema information in OODBMS which serves all of the above mentioned purposes. In this paper, we put the focus on the type lattices of the schemes.

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 section 3 a concept for the derivation of external types 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 [2].

2. A BRIEF TOUR OF THE DATA MODEL

In this section we sketch our data model just to the extent necessary for elaborating our ideas. The data model is closely tied to a schema definition language and a query language. While we provide some examples of the schema definition language, the query language is beyond the scope of this paper.

A schema in our data model consists of definitions of types and classes. Types represent intensional information, they describe the structure of objects and values, in particular, which components they consist of, and the signature and implementation of the methods that can be applied to them. Types are defined in the context of an inheritance lattice for structural inheritance. If we define a type t to be a subtype of type s, we mean that the type t inherits all the structural information from its supertype. In the definition of the subtype, we can define additional components and additional methods. Components inherited from the supertype can also be overridden in the subtype, whereby we assume that the inheritance lattice forms a subtype hierarchy in the sense of covariant subtyping. This implies, that wherever an object of a certain type can be used, we can also use an object of one of its subtypes.

The class concept has been introduced in the oo-technology in the very beginning [6]. However, types and classes usually have not been distinguished resulting in a mix of structural and extensional information. In our model, we see classes from a different perspective than the more traditional approaches. For us, classes are object containers, which can include objects compatible with a ground type. The instance of a class is the object set of the class. For each object type, there may be any number of classes, including none. There is no class automatically generated when a type is defined. Operators for insertion and removal of objects, for membership tests and iteration over the object set of a class are provided.

Classes are arranged in a class hierarchy. Since subclassing means subsetting the object set of a class, an object in a certain class *C* is also in all superclasses of *C*. An object may be in any number of unrelated classes, if the ground type of the classes are compatible with the object type. The object set of a class can be defined explicitely or by a predicate. Classes are closely connected to persistence 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.

The language for the implementation of the methods is a Turing complete procedural programming language which also can contain expressions of the query language like in O_2 [3]. The query language offers generic operations for projection, selection extension, join and set operations, similar to [8]. 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.

We distinguish between the conceptual schema, which described the database from a global, uniform point of view and external schemes, which have a perspective on the database that is specific to a group of users or applications [10]. An external schema is based on a conceptual schema and explicitly states which parts of the global schema are needed. External schemes serve as a logical buffer to minimize impacts of schema changes, be it in the conceptual or external level [5].

3. DERIVATION OF EXTERNAL TYPES

3.1 Elements of the External Schema

An external schema consists of definitions of types and of views, which are derived classes. The views are constructed from elements in the conceptual schema and from components defined in the external schema themselves. Types of the conceptual schema can be used as the basis for the definition of derived types in the external schema. The views are the derivation of classes in the conceptual schema based on a query expression which specifies how the extent of a class can be computed. In the sequel we will concentrate on the aspects concerning the type derivation.

Types in the external schema are specified with an extended version of the schema definition language used to construct the types of the conceptual schema. A definition of an external type can refer to components of the conceptual schema and use them as the basis of the definitions in the external schema.

There are basically two forms of type derivation, which can be combined to construct an external type. Restriction projects on some components of a conceptual type, extension adds new components to an external type. An external type definition which was constructed using both restriction and extension can't be easily inserted in the type lattice of the conceptual schema. It is a kind of restricted subtype or extended supertype of the conceptual type it is based on. This two approaches have annoying disadvantages. The incorporation of the external type as a restricted subtype of the conceptual type would result in a loss of the subtyping property of the lattice; in this case, it is no longer possible to use an object of a subtype where an object of a supertype is expected. The notion of an extended supertype brings with it the cumbersome upward inheritance, where a supertype (the external type) inherits attributes and methods from a subtype (the original type). This problem has been discussed in the context of types of query results, e.g. in [9].

In our approach, the solution to this problematic situation is to introduce different type lattices, one for the conceptual schema and one for each external schema. The user, an application programmer or applications themselves just see the external schema and have no possibility to gain access to the conceptual schema. A suitable representation of the external schema, where references to conceptual types are resolved and the textual definitions are integrated in the external type definitions will be provided through a schema tool. The external schema description is the complete specification of the database from a users or applications point of view. The user can query the external schema, which is also a kind of interface definition of the database. The type lattice of the external schema is part of the type lattice of the application program.

3.2 Type Restriction

Restriction is the first form of constructing an external type from a conceptual one. The restriction is twofold, it concerns attributes as well as methods. Attributes can be virtually removed by means of projection, and the set of applicable methods can be confined, too. In the definition of the external type, we also declare its external supertypes, from which it inherits in the usual manner. But also, we can reference to a type in the conceptual schema, on which the external type is based.

```
EXTERNAL TYPE et
    SUPER est1,est2
    BASED ON ct (aa, ab, ma())
END et.
```

In the example, the external type *et* inherits from external types *est1* and *est2*. It is a restriction of conceptual type *ct*, where just attributes *aa* and *ab* as well as method *ma()* are taken from *ct* and all other components of *ct* got projected away.

Note, that the instances of *ct* and *et* have exactly the same representation in the memory and that an instance of *et* is an instance of *ct* as well, because it is based on exactly one conceptual object. Transferring objects between the two type lattices does not alter their object identifier, so type derivation is completely transparent to object identity. Therefore, updates can be easily mapped to the conceptual level. Attributes which were projected away in the external type must be given an appropriate default value. This can be accomplished by the type constructor method (New()). So we can assure, that no object with an incomplete value can be inserted in the DB. As a consequence of type restriction, the components which were projected away cannot be used further on. Newly defined or redefined methods cannot access restricted attributes or call restricted methods. Such components are completely hidden from the user, they cannot be referenced in applications or queries. Furthermore, in the case object generating queries, an external type is used to specify the result type of the query.

3.3. Type Extension

The second form of type derivation, type extension., allows one to add new aspects to a type. These can be the definition of new methods or the redefinition (overriding) of existing methods. This is a powerful but also simple way of information restructuring. However, no additional stored attributes can be defined.

Additional methods can serve different purposes, they can simply be used to calculate the values of computed attributes but they can also implement arbitrarily complex behavioral aspects of the object type which do not fit well in the conceptual schema, but are essential for certain applications. The redefinition of methods is the usual way of behavioral specialization.

```
EXTERNAL TYPE eet
    SUPER est1,est2
    BASED ON ct (aa, ab, ma(), md())
    mb(): some_type;
    mc(a: this_type) : eet;
    md();
END et.
```

In the example above, methods *mb()* and *mc()* are new methods not already present in the type definitions of *est1, est2* and *ct*, or redefine methods attached to those types. Method *md*() is derived from *ct*, but also overriden by the external *md()*. In the external method, we can call methods which were defined in the conceptual type the external type is based on. Such method calls must qualify the method name with the keyword *base*. This allows for external methods to be a behavioral extension of conceptual methods, as one can use *super* to refer to methods of supertypes for the same purpose.

Here again, one instance of type *eet* is based on exactly one instance of the conceptual type *ct*. Since it is not allowed to extend a type definition with stored attributes, the representation of the instances of an external extended type exactly corresponds with the representation of the objects of the conceptual type. Preservation of object identity also allows updates of instances of extended types to be propagated to the conceptual ones.

It is allowed to combine the mechanisms of restriction and extension in an arbitrary way, which opens the possibility to omit information from the conceptual schema, as well as to enrich the external schema with aspects not covered by the global conceptual schema.

3.4. Type References in the External Schema

When types are declared to be supertypes of an external type or to be the base type of an external type, it is quite clear, that the former types themselves are external types, while the latter one must be a conceptual type.

But when references to types appear in the external schema in the definition of attributes or methods of the external types, they can either be external ones, or conceptual ones. Referencing conceptual types requires some care. For instance, if we have an external type *et* based on a conceptual type *ct*, it will be the common case that *ct* is referenced in other external type definitions. But in the external schema, we generally prefer to reference *et* and to use *ct* not at all, because do not want to "open up" the conceptual schema to the user, and want to be able to make use of the aspects of the types which were defined in the external schema only.

We provide two means to substitute an external type definition for a conceptual one. The first form allows us to cover the conceptual type in the whole external schema. Whenever a globally covered conceptual type is referenced by components in the external schema, a specified external type is used implicitly. This is a convenient way to propagate the external type definitions to all referencing types.

The second possibility to reference to external types is the individual redefinition of attributes and methods. Here, just some components make use of external types, other components refer to conceptual types.

```
EXTERNAL TYPE et
        SUPER ...
        COVERS ct (aa,ab,ma())
        ac : et2;
        mc(a: that_type) : eet;
END et.
```

In this example, we specify *et* to *cover ct*, redefining *ct* in every place it occurs in a type definition in this external schema. Attribute *ac* is defined to be of type *et2*, thereby overriding the original (conceptual) type of *ac*.

3.5. Abstracting Inheritance

With derivation, we introduced a third kind of object structuring dimension. The concept of aggregation describes that the domain of an attribute of a type is another type. (Structural) inheritance allows a type to be a specialization of other types, using additions and substitutions of components to allow reuse of existing structural and behavioral information.

Derivation of types is orthogonal to the aggregation hierarchy, and differs notably from the inheritance hierarchy. It allows to define new types by restriction, extension and redefinition of conceptual types. So it is more powerful than structural inheritance, since it unveils greater flexibility in the referencing process. In particular, it is possible to use just parts of the definition of a conceptual type, or to define multiple different external types for one conceptual type.

The main difference between structural inheritance and derivation is the aim and purpose of the two concepts. Inheritance is a way to facilitate reuse, derivation is a means to achieve the layered model and schema architecture.

3.6. Method Resolution

A type specifies which attributes and methods are defined in an object, and what piece of code gets executed when a method is invoked. In the conceptual schema a method *mc* is defined in conceptual type *ct* if it is directly defined in *ct*, or if there is at least one (conceptual) supertype of *ct*, where *mc* is defined (in the case of multiple inheritance, the programmer must explicitly resolve the ambiguities by redefining the methods).

The external schema introduces another possible path to acquire components. Besides direct definition and inheritance, the derivation path can be used to search for methods. To find the code for a method *me* in the external type *et*, the search procedure first looks if *me* is directly defined in *et*, then it follows the derivation through the projection list in the *bases on clause* of *et*. If *me* could neither be found directly in *et* nor in this clause, the search process looks for it in the external inheritance hierarchy of *et*.

We define two different execution contexts, an external one and a conceptual one. As the flow of control switches between the two contexts, different visibility rules are valid. Initially, the flow of control is in the external context. When we call a method which was defined in the conceptual schema, we change to the conceptual context, returning to the external context when the method returns.

In the conceptual context, methods cannot access any information from the external schema, but all attributes and methods defined in the conceptual schema can potentially be used. In the external context, only attributes and methods of the conceptual schema which were not restricted in the external schema can be used in addition to the extended components of the external schema itself. So a call of a conceptually defined method is a controlled way to "open up" the whole conceptual schema and to "close" it again.

4. CONCLUSION

We presented a 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 schema specification.

In this paper we could only sketch the basic ideas of this view concept. More advanced and detailed topics like view updates, schema consistency, etc. had to be skipped. We are working on an exact definition of the schema definition language together with a formalization of the semantics of the referencing mechanism, and the consistency of external schema.

REFERENCES

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

[2] 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

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

[4] 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.

[5] M.Dobrovnik, J.Eder: "View Concepts for Object-Oriented Databases." To appear in: G. Lasker (ed.): *Proc. 4th Intl. Symposium on Systems Research, Informatics and Cybernetics*, Baden-Baden, 1993

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

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

[8] 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, Ulm, 1992.

[9] 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.

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