Viewpoints in Object Database Systems

Elisa Bertino
Dipartimento di Scienze
dell'Informazione
Università di Milano
Via Comelico, 39 20133 Milano, Italy
bertino@hermes.mc.dsi.unimi.it

Giovanna Guerrini
Dipartimento di Informatica e
Scienze dell'Informazione
Università di Genova
Via Dodecaneso, 35 16146 Genova, Italy
guerrini@disi.unige.it

Abstract

In this paper we show how an object-oriented view mechanism can be exploited to allow different applications to see the same database according to different viewpoints. Each schema view is obtained by restructuring the original schema, by hiding, adding or redefining features. The basic characteristics of the proposed view mechanism are sketched and some open issues are discussed.

1 Introduction

Views are an integral component of the ANSI three level schema architecture standard, upon which the construction and use of relational database systems is centered. The schema architecture consists of the *storage schema*, describing the storage structures for a database, the *conceptual schema*, describing the logical model of the integrated database, and the *external schema*, describing the derived views of the conceptual schema for particular users or applications. Whereas foundations for the storage schema and the conceptual schema have been established for object-oriented database systems, views have only being revisited for applicability in object-oriented database systems. A foundation for the external schema will complete the development of a three-level schema architecture for object-oriented database systems, comparable to that for relational database systems.

In [7] we have proposed a view mechanism for object-oriented databases. In this paper, we show how that view mechanism is well-suited to support external schemas, that is, for providing different applications with a customized view of data. This aspect is crucial since a database is not a stand-alone entity, rather it is usually part of a complex software system, consisting of different applications, each manipulating different sets of data. Moreover, in advanced data management applications there is the need of schema versioning mechanisms and dynamic schema evolution. Schema versioning mechanisms can be very useful when dealing with schema evolution in order to provide backward compatibility towards application programs. Both those functions can be accomplished by schema view mechanisms. In particular, we discuss how from a single base schema different schema views can be derived, and how a single object database can be accessed and manipulated through different schema views. Finally, we briefly discuss some open research issues. We assume the reader to be familiar with the basics of the object-oriented data model and in particular with notions such as object, class, and ISA (inheritance) hierarchy. A survey of those concepts can be found in [3].

The paper is organized as follows. Section 2 briefly introduces the main features of the proposed view mechanism. Section 3 discusses how to provide different applications with a customized view of the database; this section also briefly addresses issues related to method dispatching, and to the schema closure property. Finally, Section 4 concludes the paper by discussing some open issues, while Appendix A contains an example.

2 A view mechanism for object-oriented databases

In our view model, two levels can be devised: views and schema views. Views are virtual classes and can be used in any context in which classes can be used; schema views provide the capability of restructuring a schema so that it meets the need of specific applications. A schema view is a virtual schema, that is, a schema which consists of views rather than of classes. Thus, our model supports the definition of external schemas whose properties are identical to those of a base schema, so that it is possible to develop object-oriented applications on an external schema. A schema view encapsulates indeed a set of related view definitions. The database administrator initially defines a base schema on which the object database is created. A schema view can be derived from either the base schema or from another schema view.

A basic characteristic of the view model we have defined is the support for both object-generating and object-preserving views. Two distinct kinds of views can be identified:

- *object-preserving views*: they are views that only extract objects from existing classes; the instances of these views can be identified by the identifiers of the extracted base objects.
- *object-generating views*: they are views creating new objects; the instances of these views must be identified by newly generated object identifiers.

Most existing object-oriented view mechanisms [10, 13, 14, 15] only consider object-preserving views. Thus, views only provide different views of existing objects. This approach is particularly useful for supporting objects with multiple interfaces and context-dependent behavior. In this sense, object-preserving views are similar to *roles* [2, 11]. However, this kind of views is not powerful enough for supporting all kinds of database reorganization. By contrast, object-generating views are proposed in [4] for supporting schema evolution, so that the evaluation of a query always returns new objects. In our model, we consider both object-preserving and object-gener-ating views. Thus, when the view query evaluation involves the creation of view instances in order to populate the view (for example, if the view query contains a join operation), new oids are generated. By contrast, when views are populated by extracting existing objects from a class or a view, possibly, modifying their structure and behavior (for example, is the view query is a selection or projection operation) the view instances preserve the identifiers of base objects, instead of generating new objects. Supporting object-preserving views requires that an object instance of a base class can also be an instance of all those views whose query is satisfied by the object. Thus, the use of the same identifier for denoting an object which is instance of both a class and a view implies that references to this object can only be solved by taking into account the *context* of the reference.

With respect to the problem of integrating views in the existing class inheritance hierarchy, the solution adopted in our model is to extend the schema with a new dimension: the *view derivation relationship* between a base class and a view, whose semantic is "the view is derived from the base class"[1] [4]. Indeed, all other proposed approaches have some disadvantages. The approach of automatically positioning views in the class hierarchy by integration algorithms [1, 12, 14] may cause the inheritance hierarchy to become very large because of the introduction many intermediate classes, semantically meaningless. The approach of requiring the programmer to explicitly specify the position of the view in the class hierarchy [8] results in a mix of classes and views in the same hierarchy that may cause confusion in the user. Moreover, it may not be trivial for the programmer to correctly position the view. The view derivation relationship we consider is orthogonal to the inheritance relationship. An inheritance relationship among views is also introduced. This relationship organizes views in an ISA hierarchy, in the same way as classes are organized. The main difference between the ISA hierarchy on base classes and the one on views is that the inheritance hierarchy on views must take into account the query component of the view. This is due to the fact that a view is not explicitly populated, as a class; rather its population is derived from the population of its root classes. Thus, to ensure that the instances of a view are a subset of the instances of its superviews, the restriction must be imposed that the query of a view must be subsumed by the queries of its superviews.

Finally, our view model allows a view to add new non-derived attributes. A view may indeed hide, modify and add features to those of the classes (or views) it is derived from. While the hiding of features (which is very useful for data protection and schema simplification) as well as the addition of new methods and the modification of method implementations are obviously needed for the definition of external schemas, the addition of non-derived attributes increases the number of possible schema changes, though making the implementation more difficult. In particular, this causes view instances having an additional storage for new attributes.

[1] If a view v is derived from a a class c, c is said *root class*. Similarly, if a schema view s is derived from a schema s', s' is said *root schema* of s.

3 Using views to provide a customized view of data to different applications

According to our view mechanism an object-oriented database is organized around the notion of base schema. A base schema is, like a schema of traditional object-oriented databases, a collection of classes related by inheritance and aggregation relationships. For providing different applications with a customized view of data, a different schema view is defined for each application. A schema view is again a collection of views (that are like classes, but are virtual in nature) related by inheritance and aggregation relationships. Thus, a schema view is a collection of views, grouped together to form a subschema. A schema view, which is derived from a given base schema through our view definition language, could be considerably different from the base schema it is derived from. In particular:

- new attributes can be added, existing attributes can be hidden, the name of an attribute of a class can be changed, the domain of an attribute of a class can be changed;
- new methods can be added, existing methods can be hidden, the name of a method of a class can be changed, the signature of a method of a class can be changed, the code of a method of a class can be changed;
- new classes can be added, existing classes can be hidden, the name of a class can be changed, inheritance relationships among classes can be changed.

We impose that schema views satisfy a *closure* property, that is, one is not completely free in choosing which views are included in a schema view. In particular, schema views must be closed under the aggregation relationship, that is, if a view is included in the schema view, also the domain of each attribute as well as the components of the signature of each operation of the view must belong to the schema view. A closed schema view contains all the views referenced directly or indirectly by the schema view definition. Since we have chosen to model a schema view as a collection of views, if the domain of a property is a class of the base schema, the corresponding identity view is introduced in the schema view, thus "virtualizing" the class without modifying it.

We remark two important aspects related to our definition of schema view closure. First, since our model does not require the existence of a common superclass of all the classes of the system, the closure property does not involve the inclusion of all the superviews of the views belonging to the schema. Second, the closure of a schema does not require the inclusion of all the subviews of the views belonging to the schema. Thus, a schema view is closed with respect to aggregation hierarchies, while it is not closed with respect to inheritance hierarchies. Indeed, the schema closure must contain the essential views for a schema to be consistent. In some contexts, it seems reasonable that a schema view includes a view and it does not include some of its subviews (so that some of the subviews are hidden in the schema view). As a consequence of this choice, the database view (that is, the database seen through the schema view) may contain objects whose most specific classes and views do not belong to the schema view. Such objects are seen through the schema view as instances of the most specific view to which the object belongs, among the ones included in the schema view. Note that this will require a careful propagation of updates, since an attribute could result in having different domains depending on through which schema view it is accessed.

Globally, a database schema consists of:

- a base schema;
- a set of schema views, corresponding to external schemas, one for each different application;
- derivation relationships, both at the class and at the schema level, relating each view to the classes it is derived from and relating each schema to the schema it is derived from; the view derivation relationship may connect view and classes belonging to different schemas, thus it is part of the global database schema rather than of an individual schema view.

Coming to the data level, in our approach we consider a single database which is associated with the global schema, and which is shared by all the existing schemas. The database contains all the instances created from classes and views belonging to the global schema. With each schema view in the global schema a database view is associated. Each database view consists of the set of objects members of views in the corresponding schema view, whose states are tailored (that is, possibly restricted) to the view in the schema view in which they are classified.

Another important aspect concerning views is how object references are solved, since "different" objects can be identified by the same object identifier. Indeed, different views of the same object are allowed, depending on the *context* in which the object is considered. In our approach, the class (or view) the referenced object must be seen an instance of, is chosen, among the ones belonging to the current schema view, taking into account the context of the object reference. The context of an object reference is simply determined by the static type of the object in the expression containing the object reference. Indeed, each object reference in each expression is assigned a single static type [5]. Thus, it is possible in each expression to derive a unique context for each expression denoting an object (object reference). The context of an object reference can be derived from the types declared for the variables in the expression and from schema information.

As far as attribute access is concerned, an attribute access $e.a$ is solved by simply returning the value of attribute a for o seen as an instance of the static type of the object reference e in the considered expression, being o the object to which reference e is instantiated. Note that, for the expression containing the reference to be a legal expression, this static type must be a view identifier belonging to the schema view of the application containing the expression and that attribute a must belong to the structure of that view in that schema view.

Method dispatching may however become more complicated since if several method implementations are applicable to a method invocation, the implementation specified in the most specific class of the invocation receiver is executed, as it is the one that *most closely matches* the invocation. However, when objects belong to classes and views not related by an inheritance relationship, the choice of the method implementation that "most closely matches" the invocation is not obvious. We have addressed the problem of dispatching for objects belonging to multiple most specific classes in [5][2]. The approach to dispatching is based on the idea that each object has in each context a *preferred* class, among its most specific ones. This approach can be easily adapted to our framework. Each method invocation is dispatched choosing the implementation in the preferred class in the current context. This approach supports a context-dependent behavior, as the same method invocation may be dispatched differently, and thus may return different results and perform different updates, depending on the context where the method is invoked. This approach is based on a total ordering on views, defined consistently with the inheritance ordering, in such a way that a method invocation is dispatched by executing:

- the method in the view which is the static type of the object reference, if this view is among the most specific ones to which the object belongs;
- the minimum with respect to the considered total ordering, of a set of views that verify the following conditions:
  - they are subviews of the static type of the object reference (this ensures that no run-time type errors occur),
  - they belong to the current schema view,

  otherwise;
- the method in the root class of the view which is the static type of the object reference, if the two cases above are not able to dispatch the method (resolution in the root schema).

Note that while the static type of the object reference certainly belongs to the schema view (otherwise the expression containing the reference would not be correct), this may not be true for any subview/subclass of this type. Thus, we remark that, though we store a single object database, the objects stored in the database behave differently depending on through which schema view they are accessed. The behavior does not only depend from the static type of the object reference, but also from the schema view in which the object reference is contained.

4 Open issues

In this paper we have discussed how an object-oriented view mechanism can be used to provide customized views of data to different applications. There are several interesting open research issues. A first open issue is how to use the proposed view mechanism to provide backward compatibility of applications upon schema changes. Whereas the use of views to experiment with schema evolution has been quite well investigated [6, 9], this "complementary" use of view has been quite neglected. This aspect is also related to the

[2] Actually, in [5] we have considered only base classes, but considering views does not introduce new problems with respect to dispatching.

need of integrated versioning mechanisms and maintenance tools for the database and the application programs. Currently, whenever the database schema changes, determining which application programs are affected must be done in most cases "by hand". Moreover, if different versions of the same programs are maintained, requiring in turn different versions of the database schema, there is the need of making sure that each program version uses the correct schema version. Since in general OODBMS and software programming environments provide distinct version management mechanisms, the bridge between the two mechanisms must be filled by some "ad-hoc" programs. Note, however, that this problem has not been much investigated. A final research issue, more in line with traditional database research, is the view update problem. This problem has been extensively investigated in deductive and relational databases. It must be revisited in the object-oriented data model.

References

[1] S. Abiteboul and A. Bonner. Objects and Views. In J. Clifford and R. King, editors, *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 238–247, 1991.

[2] A. Albano, R. Bergamini, G. Ghelli, and R. Orsini. An Object Data Model with Roles. In R. Agrawal, S. Baker, and D. Bell, editors, *Proc. Nineteenth Int'l Conf. on Very Large Data Bases*, pages 39–51, 1993.

[3] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The Object-Oriented Database System Manifesto. In W. Kim et al., editors, *Proc. First Int'l Conf. on Deductive and Object-Oriented Databases*, pages 40–57, 1989.

[4] E. Bertino. A View Mechanism for Object-Oriented Databases. In M. L. Brodie and S. Ceri, editors, *Proc. Third Int'l Conf. on Extending Database Technology*, pages 136–151, 1992.

[5] E. Bertino and G. Guerrini. Objects with Multiple Most Specific Classes. In W. Olthoff, editor, *Proc. Ninth European Conference on Object-Oriented Programming*, number 952 in Lecture Notes in Computer Science, pages 102–126, Aarhus (Denmark), August 1995.

[6] P. Breche, F. Ferrandina, and M. Kuklok. Simulation of Schema Changes using Views. In N. Revell and A M. Tjoa, editors, *Proc. Sixth International Conference and Workshop on Database and Expert Systems Applications*, number 978 in Lecture Notes in Computer Science, pages 247–258, 1995.

[7] G. Guerrini, E. Bertino, B. Catania, and J. Garcia-Molina. A Formal Model of Views for Object-Oriented Database Systems. Technical Report DISI-96-2, Università di Genova, Dipartimento di Informatica e Scienze dell'Informazione, March 1996. Submitted for publication.

[8] M. Kifer, W. Kim, and Y. Sagiv. Querying Object-Oriented Databases. In M. Stonebraker, editor, *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 393–402, 1992.

[9] W. Kim and W. Kelley. On View Support in Object-Oriented Database Systems. In W. Kim, editor, *Modern Database System*, pages 108–129. ACM Press, 1995.

[10] A. Ohori and K. Tajima. A Polimorphic Calculus for Views and Object Sharing. In *Proc. of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 255–266, 1994.

[11] J. Richardson and P. Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In J. Clifford and R. King, editors, *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 298–307, 1991.

[12] E.A. Rundensteiner. A Methodology for Supporting Multiples Views in Object-Oriented Databases. In *Proc. Eighteenth Int'l Conf. on Very Large Data Bases*, pages 187–198, 1992.

[13] J. Schilling and P. Sweeney. Three Steps to Views: Extending the Object-Oriented Paradigm. In *Proc. Fourth Int'l Conf. on Object-Oriented Programming: Systems, Languages, and Applications*, pages 353–361, 1989.

[14] M. Scholl, C. Laasch, and M. Tresch. Views in Object-Oriented Databases. In *Proc. Second International Workshop on Foundations of Models and Languages for Data and Objects*, pages 37–58, 1990.

[15] M. Staudt, M. Jarke, M. Jeusfeld, and H. Nissen. Query Classes. In S. Tsur, S. Ceri, and K. Tanaka, editors, *Proc. Third Int'l Conf. on Deductive and Object-Oriented Databases*, number 760 in Lecture Notes in Computer Science, pages 283–295, 1993.

A An example

Figure 1 shows an example of base schema, and two corresponding schema views derived from it. The schema view v_1 only consists of a view of class `Person` and of a view of class `Employee`, related by an inheritance relationship. In the view of `Person`, attributes `Father` and `Mother` have been removed, whereas attribute `Address` has been replaced by the nested attributes `Street` and `ZIPCode`. In the view of `Employee`, attribute `Company` has been replaced by the nested attribute `CompanyN`, holding the name of the company, while attribute `Manager` has been replaced by the multi-valued attribute `Dependents`; finally, the derived attribute `Monthlysal` has been added.

The schema view v_2 contains an object-generating view `Family`, relating the persons composing a family and the apartment in which the family lives. A view of class `Person` and a view of class `Apartment` are also included in the schema view. In the view of `Person` attributes `Address`, `Father`, and `Mother` have been removed, while an additional non-derived attribute `Birthday` has been introduced. In the view of `Apartment` attribute `Owner` has been removed. Finally, to ensure the closure of the schema view, the identity view of class `Address`, domain of attribute `Address` of the view of `Apartment`, belongs to v_2.

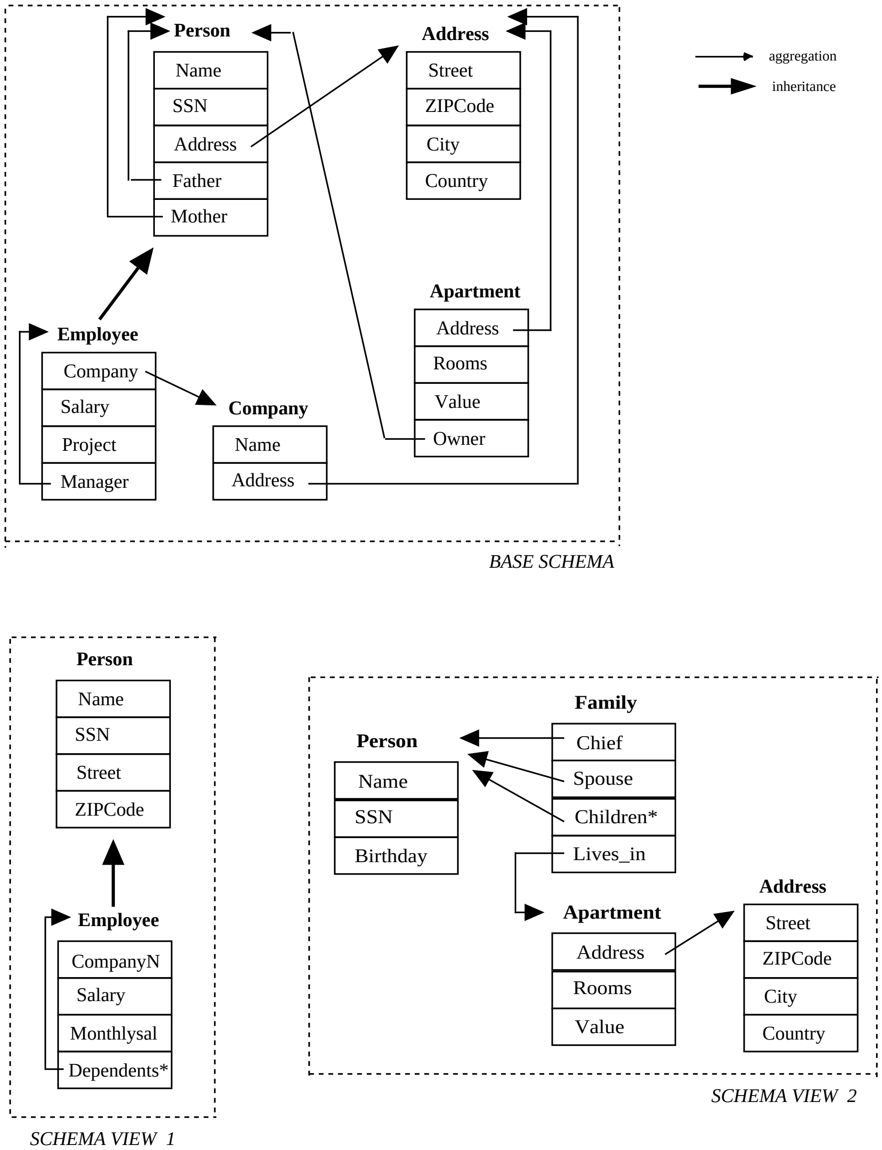

Figure 1: Example of base schema with two corresponding schema views