

# A Database Evolution Approach for Object-Oriented Databases

Awais Rashid

Computing Department, Lancaster University, Lancaster LA1 4YR, UK  
marash@comp.lancs.ac.uk

## Abstract

*This paper describes a composite evolution approach which integrates the evolution of the various types of entities in an object-oriented database into one model. The approach provides maintainers with a coherent and comprehensible view of the system and at the same time maintains change histories at a fine granularity. Links among meta-objects are implemented using dynamic relationships which are semantic constructs and first-class objects. Referential integrity is maintained by the relationships architecture reducing the evolution complexity at the meta-object level. A customisable and exchangeable instance adaptation approach is proposed. The approach is based on separating the instance adaptation code from class versions using aspects, abstractions used in Aspect-Oriented Programming to localise crosscutting concerns. A high level object-oriented model offering transparent access to the proposed evolution functionality is provided.*

## 1. Introduction

The conceptual structure of an object-oriented database may not remain constant and may vary to a large extent [24]. The need for these variations (evolution) arises due to a variety of reasons e.g.:

- to correct mistakes in the database design
- to add new features during incremental design
- to reflect changes in the structure of the real world artefacts modelled in the database

Therefore, like any other database application object database applications are subject to evolution.

However, in object-oriented databases, support for evolution is a critical requirement since it is characteristic of complex applications (e.g. computer-aided design and manufacturing, and office information systems) for which they provide support. Due to the underlying rich data model (in contrast with conventional data intensive record processing applications) these applications require dynamic modifications to both the data residing within the database and the way the data has been modelled i.e. both the objects residing within the database and the *schema* of

the database are subject to change. Furthermore, there is a requirement to keep track of the change in case it needs to be reverted.

An object-oriented database management system needs not only traditional database functionality but also the ability to dynamically evolve, through various versions, both the objects and the class definitions. This paper describes an approach – based on the separation of concerns principle – for dynamic evolution in object-oriented databases. The approach has been implemented to provide evolution support in SADES: a Semi-Autonomous Database Evolution System [20]. The next section highlights the shortcomings of existing approaches. This is followed by a discussion of the proposed approach.

## 2. Shortcomings of existing approaches

Historically, the database community has employed three fundamental techniques for modifying the conceptual structure of an object-oriented database, namely:

- **schema evolution** e.g. [2] [13], where the database has one logical schema to which class definition and class hierarchy modifications are applied
- **class versioning** e.g. [3] [12] [25], which keeps different versions of each type and binds instances to a specific version of the type
- **schema versioning** e.g. [10] [14], which allows several versions of one logical schema to be created and manipulated independently of each other

In addition, for managing the evolution of objects residing in the database a number of **object versioning** strategies e.g. [8] [23] have been employed.

The above approaches tend to concentrate on a particular aspect of evolution instead of considering evolution as a process which affects the whole database. Approaches in the schema evolution category provide maintainers with a coherent and comprehensible view of the system. They normally employ user-defined or default transformation functions [5] together with screening [5] (in case of a deferred update) in order to bring existing objects in line with the modified schema. However, essential information about historical changes is lost. This

is a major drawback as a change might need to be reverted.

Class versioning approaches, on the other hand, maintain information about historical class changes. There is no need to change existing objects to reflect schema changes. Error handlers [25] or update/backdate methods [12] can be employed to access objects of a particular class using different versions of the class. However, as the number of versions for each class grows the schema tends to become quite complex, making maintenance difficult. It is also hard to obtain a coherent view of the system in the presence of a large number of class versions.

Schema versioning alleviates the shortcomings of schema evolution and class versioning by providing a coherent view of the conceptual structure of the database, while at the same time maintaining information about historical changes to the schema. The historical information is, however, not available at a finer granularity as in class versioning. Although particularly useful for maintaining forward and backward compatibility with existing applications, schema versioning can be very expensive in terms of space usage, especially in the presence of a large number of schema versions. Minor class changes also lead to the creation of a new schema version which is a significant overhead. Approaches such as [14] counter this problem by simulating schema versions using object-oriented view techniques. However, information about historical changes is again available at a coarser granularity and additional overhead is introduced due to view maintenance.

With the exception of AVANCE [3] and [10], existing systems consider object versioning a separate activity. It is not integrated within the evolution model. Consequently an experienced application developer aiming at managing evolution of both data and meta-data faces an intellectual barrier as s/he has to interact with two separate evolution models.

With the exception of Iris [6], the meta-object hierarchy in existing systems is based on static references. As a result the code handling the connections among the meta-objects is spread across the meta-object space. This makes implementation of dynamic evolution features cumbersome as an additional level of complexity is introduced due to the evolution problem moving into the meta-object level. Furthermore, hierarchy graph edges have to be labelled in order to correctly determine the particular links represented by them. This introduces additional overhead when new edges are introduced or existing ones removed or modified.

The instance adaptation strategy in existing systems is fixed. It is not possible to customise it for a specific evolution scenario or exchange it with an entirely different strategy over the lifetime of the database. The latter might be required due to the availability of a more sophisticated,

efficient strategy. The instance adaptation routines in these systems are spread across the various classes or class versions. This makes maintenance expensive as changes are not localised.

Although existing systems offer reasonable evolution features within their respective categories, access to these features is not transparent i.e. the change specification language is not orthogonal to the programming language and unnecessary schema implementation details are not hidden from the maintainers. The application programmer has to interact with the lower level object-oriented model of the particular system: the implementation details of the schema. A high level object-oriented model offering abstractions to hide the schema implementation details in order to provide transparent access to the evolution functionality of the OODBMS is not available.

### 3. A composite evolution approach

As discussed in section 2, existing approaches tend to concentrate on a particular aspect of evolution. The proposed approach, in contrast, views evolution as a process which affects the whole database. It, therefore, focuses on "Database Evolution" and superimposes schema evolution on class versioning [20] offering a composite evolution approach providing support for:

- Class hierarchy evolution
- Class versioning
- Object versioning

A composite evolution model [20] based on version derivation graphs [11] is proposed and used as a basis to define an evolution taxonomy [15]. In contrast with existing taxonomies e.g. [2] the proposed evolution primitives are based on an evolution model and not the data model of the particular object database management system. The taxonomy can, therefore, be supported by any DBMS supporting the composite evolution model. The composite evolution model provides a coherent and comprehensible view of the system making maintenance easier. The model forms the basis of the SADES conceptual schema.

### 4. Evolution using dynamic relationships

[7] characterises relationships among classes as static since these are fixed at compile-time. Relationships among instances are comparatively dynamic in nature and can be changed at run-time. The viewpoint in this paper differs from that presented by [7]. Relationships among classes do not need to be fixed at compile-time. These can be dynamic in nature and can be changed at run-time. Therefore, the schema of an object-oriented database can be dynamically modified if the various meta-objects (classes, etc.) that form the schema are interconnected

through dynamic relationships. Dynamic schema changes can be made by dynamically modifying the various relationships in which the meta-objects participate. These can be the *derives-from/inherited-by* relationships between classes or the *defines/defined-in* relationships between classes and their members. If relationships exist among meta-objects and objects they can be used to propagate schema changes to the affected objects.

The proposed evolution mechanism employs a dynamic relationships approach [19] based on the observation that three types of entities exist within an object-oriented database. These are objects, meta-objects and meta-classes. Relationships are treated in a uniform fashion regardless of the type of participating entities and the type of the relationship (aggregation or association). The dynamic relationships approach is used to implement the composite evolution model and the various evolution primitives in the taxonomy.

The use of dynamic relationships to achieve dynamic evolution serves a two-fold purpose. First, a coherent view of the conceptual structure of the database is provided making maintenance easier. Second, the information about the connections among the various meta-objects is separated and encapsulated in the relationship constructs. Since relationships can be dynamically introduced, removed or modified the evolution framework is more extensible and maintainable. The introduction, removal or modification of meta-classes only requires introduction, removal or modification of the various semantic relationships with referential integrity managed by the system. Propagation patterns and semantics of relationships can also be modified in an independent fashion without a significant impact on the evolution framework.

The dynamic relationships approach has been implemented as a layer on top of the commercially available object database management system Jasmine [1] using ODQL (the Jasmine database language) and C/C++. The functionality exposed by this layer has been employed to implement the SADES conceptual schema and evolution primitives based on the evolution model and taxonomy outlined in section 3.

## 5. Flexible instance adaptation

The proposed evolution approach observes that the instance adaptation strategy in an object database evolution system crosscuts the various class versions. Special abstractions called *aspects* used by Aspect Oriented Programming [9] to localise crosscutting concerns are employed to separate the instance adaptation strategy from the class versions [22]. This provides greater flexibility and maintainability as changes to the instance adaptation code are localised and it is possible to

customise the instance adaptation strategy or seamlessly move to an entirely different one if required.

The dynamic relationships based implementation of SADES outlined in section 4 has provided support for seamlessly extending the system with a meta-class *Aspect* and integrating the aspect-oriented instance adaptation approach into the system [18]. The instance adaptation aspects are specified using a declarative aspect-oriented language and dynamically woven using a weaver (a tool to compose aspects and entities crosscut by them) [9] supporting run-time and persistent aspects. A description of using two different instance adaptation strategies: error handlers and update/backdate methods and switching between them in a flexible, cost-effective fashion in SADES can be found in [15] [22].

## 6. Transparent access to evolution functionality

With the increasing provision by the ODMG standard [4] and commercial OODBMS products for transparent access to the traditional database functionality of an OODBMS, there is an urgent need to provide database application programmers with transparent access to advanced functionality such as dynamic evolution. The proposed evolution mechanism offers a transparent API based on a high-level object-oriented model to provide access to the evolution functionality offered by the composite evolution model (implemented using dynamic relationships) [21]. The application programmer interacts with the high-level model instead of interacting with the lower level model of the particular DBMS.

## 7. Conclusion

This paper has summarised the work presented in the author's PhD thesis [15]. The discussion in this paper and the thesis has highlighted the need for comprehensible but rich evolution models for object-oriented databases and demonstrated their feasibility. It has also identified various crosscutting concerns introducing additional complexity into evolution frameworks and provided a strong argument for applying the separation of concerns principle to localise changes and improve the maintainability and extensibility of evolution frameworks. The notion of maintainable and extensible evolution frameworks is a novel contribution and takes research in object database evolution to a new dimension. The proposed concepts not only consider evolution of the entities residing in the database but also management and localisation of changes to the evolution framework itself. This is in direct contrast with existing evolution systems which do not take into account the evolution of the

database management system (and hence the evolution framework).

The use of separation of concerns for object database evolution has also led to the concept of *Aspect-Oriented Databases* [16] [18]. In an aspect-oriented database separation of concerns is explicitly taken into account. Various crosscutting concerns are separated which localises changes and improves the maintainability and extensibility of the system. Such databases also provide storage facilities for aspects in application programs [17].

In summary the approach described has provided a new perspective on object database evolution and strong grounds for development of maintainable and extensible databases in general and evolution frameworks in particular.

## 8. References

- [1] *The Jasmine Documentation*, 1996-1998 ed: Computer Associates International, Inc. & Fujitsu Limited, 1996.
- [2] J. Banerjee, H.-T. Chou, J. F. Garza, W. Kim, D. Woelk, and N. Ballou, "Data Model Issues for Object-Oriented Applications", *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, pp. 3-26, 1987.
- [3] A. Bjornerstedt and C. Hulthen, "Version Control in an Object-Oriented Architecture", in *Object-Oriented Concepts, Databases, and Applications*, W. Kim, Lochovsky, F. H., Ed., 1989, pp. 451-485.
- [4] R. G. G. Cattell, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russel, O. Schadow, T. Stenienda, and F. Velez, *The Object Data Standard: ODMG 3.0*: Morgan Kaufmann, 2000.
- [5] F. Ferrandina, T. Meyer, R. Zicari, and G. Ferran, "Schema and Database Evolution in the O2 Object Database System", VLDB Conf., 1995, Morgan Kaufmann, pp. 170-181.
- [6] D. H. Fishman, et al., "Iris: An Object Oriented Database Management System", *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, pp. 48-69, 1987.
- [7] E. Gamma, et al., *Design Patterns - Elements of Reusable Object-Oriented Software*: Addison Wesley, 1995.
- [8] R. H. Katz, "Toward a Unified Framework for Version Modeling in Engineering Databases", *ACM Computing Surveys*, Vol. 22, No. 4, pp. 375-408, 1990.
- [9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin, "Aspect-Oriented Programming", ECOOP, 1997, Springer-Verlag, LNCS 1241
- [10] W. Kim and H.-T. Chou, "Versions of Schema for Object-Oriented Databases", VLDB Conf., 1988, Morgan Kaufmann, pp. 148-159.
- [11] M. E. S. Loomis, "Object Versioning", *Journal of Object Oriented Programming*, No., pp. 40-43, 1992.
- [12] S. Monk and I. Sommerville, "Schema Evolution in OODBs Using Class Versioning", *ACM SIGMOD Record*, Vol. 22, No. 3, pp. 16-22, 1993.
- [13] R. J. Peters and M. T. Ozsu, "An Axiomatic Model of Dynamic Schema Evolution in Objectbase Systems", *ACM Transactions on Database Systems*, Vol. 22, No. 1, pp. 75-114, 1997.
- [14] Y.-G. Ra and E. A. Rundensteiner, "A Transparent Schema-Evolution System Based on Object-Oriented View Technology", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 4, pp. 600-624, 1997.
- [15] A. Rashid, "A Database Evolution Approach for Object-Oriented Databases", PhD Thesis, *Computing Department*: Lancaster University, UK, 2000.
- [16] A. Rashid, "A Hybrid Approach to Separation of Concerns: The Story of SADES", 3rd International Conference on Meta-Level Architectures and Separation of Concerns Reflection 2001 (To Appear)
- [17] A. Rashid, "On to Aspect Persistence", 2nd International Symposium on Generative and Component-based Software Engineering, 2000, Springer-Verlag, LNCS 2177
- [18] A. Rashid and E. Pulvermueller, "From Object-Oriented to Aspect-Oriented Databases", DEXA Conf. 2000, Springer-Verlag, LNCS 1873, pp. 125-134.
- [19] A. Rashid and P. Sawyer, "Dynamic Relationships in Object Oriented Databases: A Uniform Approach", DEXA Conf., 1999, LNCS 1677, pp. 26-35.
- [20] A. Rashid and P. Sawyer, "Facilitating Virtual Representation of CAD Data through a Learning Based Approach to Conceptual Database Evolution Employing Direct Instance Sharing", DEXA Conf., 1998, Springer-Verlag, LNCS 1460, pp. 384-393.
- [21] A. Rashid and P. Sawyer, "Transparent Dynamic Database Evolution from Java", *L' Object*, Vol. 6, No. 3, pp. 373-386, 2000.
- [22] A. Rashid, P. Sawyer, and E. Pulvermueller, "A Flexible Approach for Instance Adaptation during Class Versioning", ECOOP 2000 Symposium on Objects and Databases, 2000, Springer-Verlag, LNCS 1944, pp. 101-113.
- [23] I. Santoyridis, T. W. Carnduff, W. A. Gray, and J. Miles, "An Object Versioning System to Support Collaborative Design within a Concurrent Engineering", 15th British National Conference on Databases BNCOD, 1997, pp. 184-199.
- [24] D. Sjoberg, "Quantifying Schema Evolution", *Information and Software Technology*, Vol. 35, No. 1, pp. 35-44, 1993.
- [25] A. H. Skarra and S. B. Zdonik, "The Management of Changing Types in an Object-Oriented Database", 1st OOPSLA Conference, 1986, pp. 483-495.