Modelling inheritance, composition and relationship links between objects, object versions and class versions

E. Andonoff*, G. Hubert, A. Le Parc**, G. Zurfluh***

* Lab. CERISS, Univ. Toulouse I, 31042 Toulouse Cédex, France
Email : ando@cix.cict.fr

** Lab. IRIT pôle SIG, Univ. Toulouse III, 31062 Toulouse Cédex, France
Email : {ando,hubert,leparc,zurfluh}@irit.fr

Abstract. This paper presents a conceptual object-oriented model which allows to describe, in a unified framework, objects, object versions and class versions. Three kinds of classes are used for such a modeling: object classes, version classes and versionable classes. This paper approaches, in greater details, the outcomes of representing links between these different kinds of classes. The considered links are inheritance, composition and relationship links. Most of system managing object versions and/or class versions only partially approach this problem.

keywords. Object classes. Version classes. Versionnable classes. Inheritance. Composition. Relationship.

1 Introduction

The concept of version was introduced to describe the evolution of real world entities along time. The different states of entities are kept and correspond to their different versions. The concept of version is important in computer aided design, technical documentation or software engineering fields where managed data are time-dependent [2].

Many current database systems allow to manage versions. In these systems, versions are investigated at two abstraction levels: the object level and the class level. Some systems have studied versions at only one abstraction level. For example, Sherpa [14], O2 [20] or CloSQL [15] have studied this concept at the class level while Etic [9] has investigated it at the object level. Other systems, such as Orion [7] [12], Encore [19], Iris [3], Avance [4], OVM [10] or Presage [18] have studied versions at the two abstraction levels. But these studies are often (except Presage) carried out without analysing the outcomes of both managemement of class versions and object versions. Moreover, these studies partially deal with (or bypass) the study of links and cardinalities between versions: composition is partially approached while there is no work about inheritance and relationship.

On the other hand, current database design methods such as OMT [17], OOA [6], OOM [16], O* [5] do not propose solutions to model versions in their conceptual data models. Now, version modeling is an activity recovered from the conceptual level [11]. Indeed, to exactly describe the real world, a designer must be able to tell if an entity may evolve or if an entity class may evolve. So, he must have tools to model object and class versions in the conceptual data models he defines.

In this paper, we present a conceptual object model intended for object-oriented database modeling. This model enables us to describe, in a unified framework, objects, object versions and class versions. Three kinds of classes are used for such a modeling: object classes, version classes and versionable classes. Object class instances are objects of which one keeps only the last state. The instances of version classes are object versions where only the value evolves whereas the instances of versionable classes are object versions whose value and schema evolve (the different significant states are kept). This paper investigates, in greater details, the outcomes of modeling links between these different kinds of classes on objects and object versions. The considered links are inheritance, composition and relationship links. Moreover, operations on objects and classes are also discussed in this study.

This paper is organised as follows. Section 2 presents the main concepts of version and particularly the concepts of object and class versions. Section 3 describes the conceptual object model we have defined; it shows how we represent objects, object versions and class versions using three kinds of classes. Sections 4, 5 and 6 approach the outcomes of modeling inheritance, composition and relationship links between these various kinds of classes. Section 7 is the conclusion.

2 The Concepts

The version concept is studied in a unified way at two abstraction levels which are the object level and the class level.

2.1 Versions

In the real world, an entity has characteristics which may evolve during its life cycle: the entity has different successive states.

In object-oriented databases, a real world entity is described by a unique object. This object has a schema (i.e. a set of attributes and methods) and a value. The schema and the value describe the last entity state.

In a version context, an entity is not described by a unique object but by a set of objects (versions): it is possible to manage several entity states and not only the last one. A version corresponds to one of the entity states. The entity versions are linked by a derivation link; they constitute a version derivation hierarchy [11].

An entity class is described by a set of version hierarchies; each entity is described by only one hierarchy.

When created, an entity is described by only one version (called root version). The definition of every new entity version is done by derivation from a previous version. Such versions are called derived versions [11] (e.g. E1.v1 is a derived version from

E1.v0). Several versions may be derived from the same previous version. Such versions are called alternatives [11] (e.g. E1.v2 and E1.v3 are alternatives derived from E1.v1).

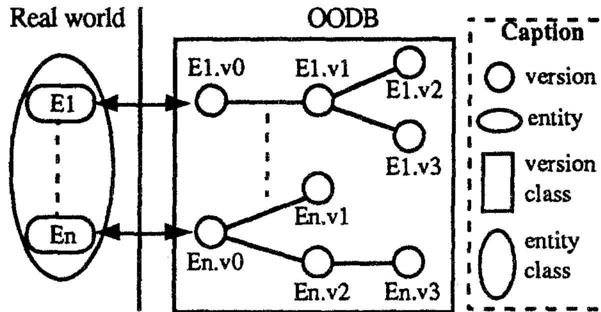


Fig. 1. Representing entities with versions

A version is either frozen or working. A frozen version describes a significant and final state of an entity. A frozen version may be deleted but not updated. To describe a new state of this entity, we have to derive a new version (from the frozen one). A working version is a version that temporarily describes one of the entity states. It may be deleted and updated to describe a next entity state. The previous state is lost for the benefit of the next state.

The default version of an entity [7] is a version pointed out by the database designer as the most representative version of the entity. It is unique for each entity and may be chosen among the set of frozen or working versions.

2.2 Object Versions and Class Versions

In object-oriented databases, versions may be considered at two abstraction levels: the object level and the class level. Evolution of objects may found expression in either value evolution or schema evolution. If an entity evolution is described by a set of versions having the same schema (there is only value evolution), one talks about object version. If an entity evolution is described by a set of versions which do not have the same schema (there is value and schema evolution), one talks about class version: these different versions belong to different classes. Value evolution consists in updating (in a partial or a total way) the attribute values of the considered object version. Schema evolution consists in adding new attributes or new methods, or in updating or deleting attributes or methods already defined, of the considered class version.

Class or object evolution is realized by deriving an object or a class version. Thus, derived or alternative object versions or class versions are created. These versions, like entity versions, are linked by a derivation relationship; they constitute a version derivation hierarchy (for objects and classes). The frozen, working and default version notions are available for object and class versions. The figure below illustrates object and class versions concepts.

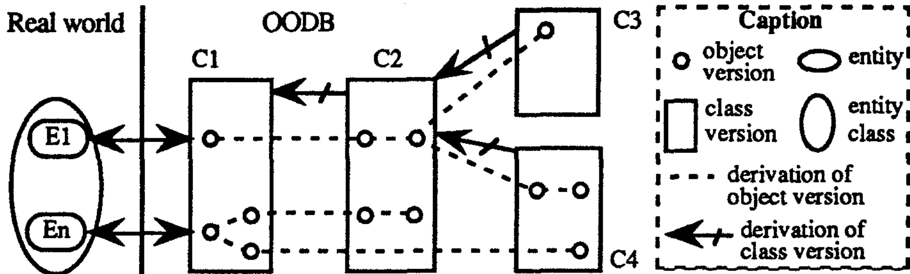


Fig. 2. Object versions and class versions

3 The Model

The data model we define is a conceptual model which takes its inspiration from the OMT object model [17]. It enables to describe objects, object versions and class versions.

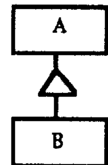
3.1 Object Classes and Links

Class is the unique tool for modeling real world entities. A class gathers a set of objects having the same schema. The class schema describes the structure and behaviour of its objects. The structure is represented by a set of (monovalued or multivalued) attributes whose domain is a predefined class (integer, real...). Behaviour is described by a set of methods described by their signature. Classes may be linked by three kinds of links : inheritance links, composition links and relationship links.

3.2 Inheritance Links

Inheritance models *is-a* link between objects. It allows to gather the common properties (attributes and methods) of several classes, called subclasses, in a more general class, called superclass. Inheritance is the mechanism allowing to transfer properties from superclass to subclasses.

The model retains a specialization inheritance [1]: subclasses are described defining new properties or redefining inherited properties. The inheritance hierarchy of a class consists of the class itself and all the classes belonging to the inheritance hierarchy of its subclasses. Inheritance link is shown on opposite figure.

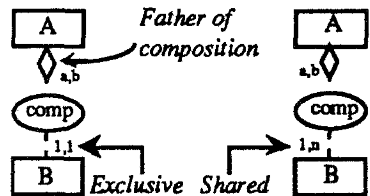


3.3 Composition Links

Composition models *is-part-of* link between objects. It is a basic modeling tool to describe complex objects [13]. A composition link is defined between two classes when life cycles of objects belonging to the two classes are dependent [5] (e.g. the creation of an object in a class causes the creation of an object in the other class). Composition links are either exclusive or shared. If a link is an exclusive one, a component object can only be a part of one composite object. If the link is a shared one, a component object can be a part of several composite objects.

At last, the cardinality of a composition link is indicated in a conventional way: on the one hand a,b (a in {0,1}, b in {1,n}), and, on the other hand c,d (c=1, d in {1,n}).

A composition link is shown on opposite figure. A diamond indicates the composite class. The cardinality for the component class indicates if the link is exclusive (1,1) or shared (1,n). We can note that 0 is not authorized for the component class [13].



Composition links were studied in Orion. [13] introduces the dependent independent, exclusive and shared notions. The composition links we model correspond to the dependent exclusive and dependent shared composite references of Orion.

3.4 Relationship Links

Relationship models *is-linked-to* link between objects. A relationship link is defined between two classes when life cycles of objects belonging to the two classes are independent [5]. The cardinality of a relationship link is indicated as before.

Relationship links are shown as follows:

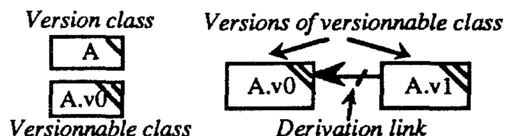
The relationship links we model correspond to the relationship links defined in OMT, OOA, OOM or O*, and to the independent exclusive and independent shared composite references defined in Orion.

3.5 Model Extensions

The model is extended to integrate the notions of object and class versions. In addition to object classes (whose instances are objects), it allows to model two other kinds of classes (whose instances are object versions): version classes and versionable classes.

Instances of version classes are object versions having only value evolution whereas instances of versionable classes are object versions having value and schema evolution. Version classes have a schema of which one keeps only the last state whereas versionable classes have a schema that may evolve (different significant and final states are kept). The different class versions describing schema evolution of a versionable class are linked by a derivation link. Object evolution is represented by a hierarchy of object versions belonging to one or more classes linked by a derivation link. We can observe that a versionable class is also a version class.

Version classes, versionable classes and derivation link between class versions of a versionable class are shown as follows:



Inheritance, composition and relationship links (previously shown between object classes) can be defined between the different kinds of classes. Different inheritance, composition and relationship cases are conceivable.

The constraints underlying these kinds of classes make some cases inconsistent or restrict the operations which can be performed on classes and their instances.

3.6 Operations

Operations which can be performed on classes (a) and their instances (b) are described in the following table :

operation / *class*	derive (a)	update (a)	create (b)	delete (b)	derive (b)	update (b)
object class	no	yes	yes	yes	no	yes
version class	no	yes (*)	yes	yes	yes	yes (*)
versionable class	yes	yes (*)	yes	yes	yes	yes (*)

Class operations allow to create a new class deriving an existing one (derive) or to modify a class schema (update). Instance operations allow to create (from scratch) a new instance (create), to delete an existing instance (delete), to create a new instance deriving an existing one (derive) or to modify the instance values (update). We can observe that the update (*) operation can only be performed to unfrozen object versions (instances of version or versionable classes) or to unfrozen versionable classes. These operations are detailed in the following sections. The links defined between classes (inheritance, composition and relationships links) are also taken into account.

4 Inheritance

We first describe the different inheritance cases and then study the instance and class operations.

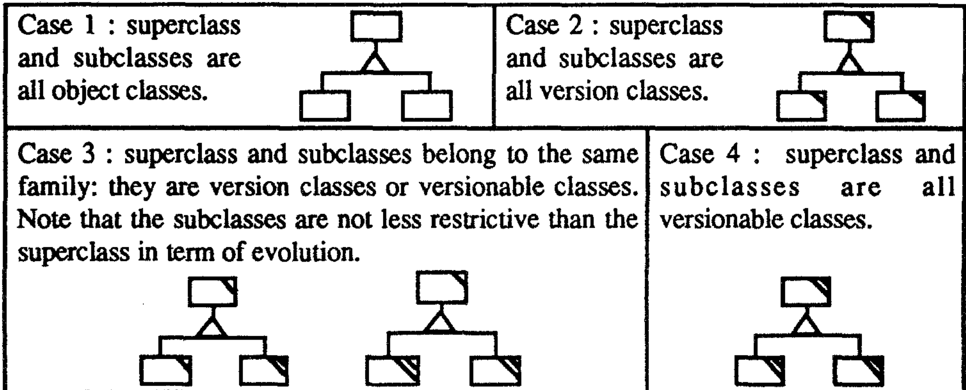
4.1 Inheritance Cases

A class belongs to one of the two following families : object or version. The object family consists of object classes. The version family consists of version and versionable classes. We can observe that a versionable class is less restrictive than a version class in term of evolution because it allows schema evolution in addition to value evolution.

We have defined two rules that indicate if an inheritance hierarchy is consistent. These rules allow to organize in inheritance hierarchies classes belonging to the same family (the instances of these classes evolve in a similar way: no evolution or value and/or schema evolution). These rules are the following :

- the subclasses and the superclass must belong to the same family, i.e. either object family or version family;
- the superclass category must not be less restrictive (in term of evolution) than its subclasses categories.

Inheritance cases checking these rules are described below. Any other case is forbidden.



4.2 Operations

Instance operations are only performed on subclasses leaf of the inheritance hierarchy (the superclasses, i.e. the classes which are not leaf of a inheritance hierarchy, do not have their own instances). These instance operations (create, delete, derive and update) are available without restriction (as described in section 3.6).

With respect to inheritance cases, class operations (derive and update) may be forbidden or performed with or without restriction:

operations / *possible cases*	derive superclass	derive subclass	update superclass	update subclass
case 1	no	no	yes (3)	yes (4)
case 2	no	no	yes (5)	yes (6)
case 3	no	yes (1)	yes (5)	yes (6)
case 4	yes	yes (2)	yes (5)	yes (6)

The following comments explain these different operations:

- (1) schema derivation is permitted in a versionable subclass. This derivation does not affect inherited attributes.
- (2) schema derivation is permitted in the versionable subclasses. If inherited attributes are updated in the subclass, then there is repercussion of this derivation on the superclass and on the other subclasses.
- (3) schema updating in a superclass causes schema updating in its subclasses. This updating is propagated on subclasses instances.
- (4) schema updating in a subclass is propagated on the superclass and the other subclasses if this modification affects inherited attributes. This updating is propagated on subclasses instances.
- (5) schema updating in a superclass is permitted if there are no frozen instances in its subclasses. In this case, the comment (3) is applicable.
- (6) schema updating in a subclass is permitted if the subclass has no frozen instances. In this case, the comment (4) is applicable.

5 Composition

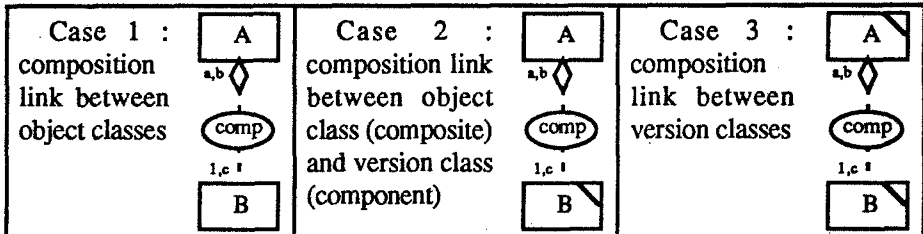
We first describe the different composition cases and then study the instance and class operations. For space limitations, we only present the composition links whose cardinalities are (1,1/1,1), (1,1/1,n), (1,n/1,1) and (1,n/1,n). These cardinalities are the most restrictive ones, and, therefore, the most interesting ones.

5.1 Composition Cases

A composition link may be defined between the three kinds of classes: object, version and versionable classes. However, only the notion of object version must be considered in composition study. Indeed, instances of version and versionable classes are object versions. So, it is useless to distinguish version classes from versionable classes.

We have defined one rule which indicates if a composition link is consistent. This rule allows to organize in composition hierarchies the classes whose instances have similar life cycles (i.e. dependent). This rule imposes not to have a composite class which is version class, and a component class which is an object class.

The composition cases checking this rule are described below. Cases 1 and 3 are studied in the literature while case 2 is never approached.

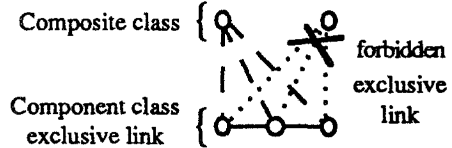


For each of the three previous cases, we study how the integrity constraints inherent in composition link cardinalities (they express the exclusive and sharing notions) are taken into account at both object level and object version level. Note that the chosen solutions limit the useless duplications of versions.

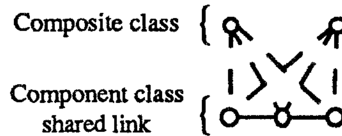
Case 1 : composition between object classes. This case is classic and is widely approached in the literature. The solution we retain is the one proposed in [13] (for more details, report to [13]).

Case 2 : composition between object class and version class. On the one hand, this case expresses a link between a composite class and one or more hierarchies of component object versions, and, on the other hand, this case expresses a link between one component object version and one (exclusive) or more (shared) composite objects.

If the composition link is exclusive, a component hierarchy is a part of only one composite object.

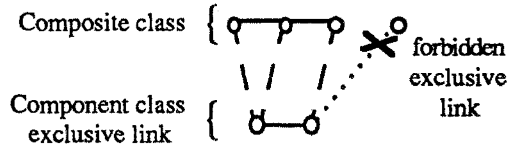


If the composition link is shared, a component hierarchy is a part of one or more composite objects.

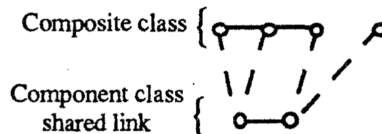


Case 3 : composition between version classes. On the one hand, this case expresses a link between a composite object version and one or more hierarchies of component object versions, and, on the other hand, this case expresses a link between a component object version and one (exclusive) or more (shared) composite object versions.

If the composition link is exclusive, a version of a component hierarchy is a part of only one composite hierarchy.



If the composition link is shared, a version of a component hierarchy is a part of one or more composite hierarchies.



This case is studied in most of system managing object versions (e.g. Presage, Orion, ...). Presage duplicates object versions. This solution allows to check the integrity constraints which are inherent in composition links; but it causes the (useless) creation of several object versions describing the same object evolution states. The Orion solution consists in connecting the derived versions of an object to the component generic object. Such a technic limits object version duplication but it imposes to dynamically compute the derived versions.

5.2 Operations

Composition expresses that composite and component instance life cycles are dependent. So, some operations performed on instances (create, delete, derive) must be studied in details. The other ones are available as described in section 3.6.

Creation. The dependency of composite and component instance life cycles imposes to create one (or more) component instance(s) when creating a composite instance [13]. Exclusivity and sharing notions and the different composition cases must also be taken into account.

Case 1 : composition between an object class A and an object class B. This case is classic. We retain the solution proposed in [13].

Case 2 : composition between an object class A and a version class B. If the composition link is monovalued for A (1,1), when a new object "a" is created in the composite class A, it must be linked to a unique hierarchy of the component class B (i.e. to all the versions of this hierarchy). If the link is multivalued for A (1,n), when an object "a" is created in the composite class A, it must be linked to one or more hierarchies of the component class B.

If the link is exclusive (1,1), the hierarchy in B must be a new one. If the link is shared (1,n), the hierarchies in B are either existing or new hierarchies.

Case 3 : composition between a version class A and a version class B. If the composition link is monovalued for A, a root version "a" of a new hierarchy created in the composite class A must be linked to a unique version "b" of a hierarchy of the component class B. If the link is multivalued, the root version "a" can be connected to one or more version "b" of the component class B. These versions belong to distinct hierarchies.

If the link is exclusive, the version "b" of the component class B is either a root version of a new hierarchy, or a derived of a leaf version of a free hierarchy. A component hierarchy is described as free when it has no leaf versions linked to a leaf version of a composite hierarchy:

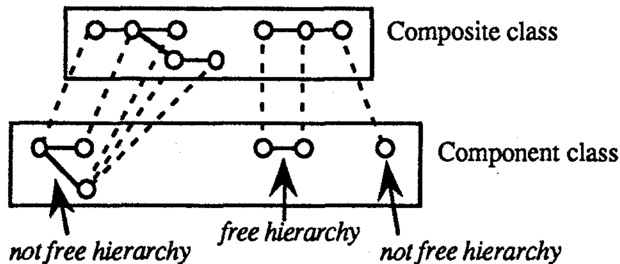


Fig. 3. Free hierarchies

If the link is shared, the versions "b" of the class B are either leaf versions of existing hierarchies, or root versions of new hierarchies.

Deletion. The dependency of composite and component instance life cycles imposes to delete one (or more) component instance(s) when deleting a composite instance [13]. Exclusivity and sharing notions and the different composition cases must also be taken into account.

Case 1 : composition between an object class A and an object class B. This case is classic. We retain the solution proposed in [13].

Case 2 : composition between an object class A and a version class B. If the composition link is exclusive, the deletion of an object "a" in the composite class A causes the deletion of all its component hierarchies in the component class B (i.e. the deletion of the versions belonging to these hierarchies). If the link is shared, the component hierarchies are only deleted if they do not compose other objects belonging to the composite class A.

Deleting a component hierarchy implies to connect the composite object to another component hierarchy (according to link cardinality).

Case 3 : composition between a version class A a version class B. If the composition link is exclusive, the deletion of a version "a" in the composite class A causes the deletion of its component versions "b" belonging to the component class B except if the component versions compose other versions ("aa") of the same composite hierarchy. On the other hand, if the link is shared, the component versions "b" are deleted only if they do not compose other versions.

The deletion of a component version causes the deletion of the corresponding composite versions which are frozen (several successive derived versions can have the same components). If the composite versions are working, they are not deleted but linked to other component versions.

Derivation. Derivation is an operation which can only be performed on instances of version classes. Only case 2 and case 3 are studied.

Case 2 : composition between an object class A and a version class B. A derived version in the component class is automatically linked to the same composite objects as the version it derives from (the previous one).

Case 3 : composition between a version class A and a version class B. If the composition link is exclusive, the derivation of a composite version causes the creation of a new version linked to one or more versions which can be :

- the same component versions as the composite version it derives from,
- derived versions of the component versions,
- versions which are roots of new hierarchies belonging to the component class,
- new versions derived from leaf versions of free component hierarchies (cf 5.2.1).

If a composition link is shared, the result of composite version derivation is a version which can also be linked to one or more leaf versions belonging to any component hierarchy.

Deriving a component version causes deriving all the linked composite versions. We can observe that this derivation does not cause the derivation of the other components of the composite object; the other (composition and relationship) links are not modified.

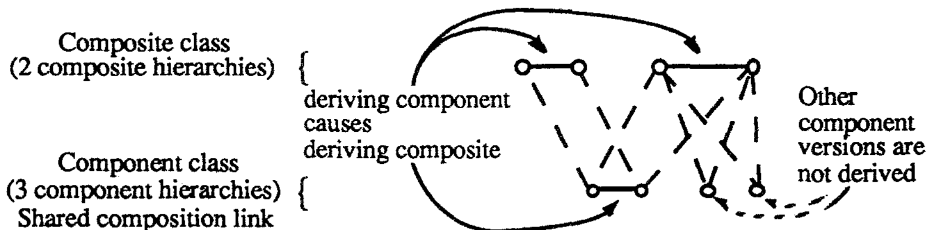


Fig. 4. Consequences of component derivation

6 Relationship

We first describe the different relationship cases and then study the instance and class operations. For space limitations, we only present the relationship links whose cardinalities are (1,1/1,1), (1,1/1,n) and (1,n/1,n). These cardinalities are the most restrictive ones, and, therefore, the most interesting ones.

6.1 Relationship Cases

A relationship link may be defined between the three kinds of classes: object, version and versionable classes. However, only the notion of object version must be considered in relationship study. Indeed, instances of version and versionable classes are object versions. So, it is useless to distinguish version classes from versionable classes. The relationship cases which must be studied are described below. Cases 1 and 3 are approached in the literature whereas case 2 is never investigated.

Case 1 : relationship link between object classes

Case 2 : relationship link between an object class and a version class

Case 3 : relationship link between version classes

For each of the three previous cases, we study how the integrity constraints inherent in relationship link cardinalities are taken into account at both object level and object version level. Note that the chosen solutions limit useless duplications of versions.

Case 1: relationship between object classes. This case is widely studied in the literature (OMT, OOA, OOM, O*, ...). It is not presented in this paper.

Case 2: relationship between an object class and a version class. On the one hand, this case expresses a link between an object and one or more hierarchies of object versions, and, on the other hand, this case expresses a link between an object version and one or more objects.

On the other hand, if the relationship link is monovalued for the object class A (1,1), an object of A must be linked to only one current hierarchy of B.

The current hierarchy for an object is the last hierarchy linked to it. If a relationship link is multivalued for the object class A (1,n), an object must be linked to one or more hierarchies of B.

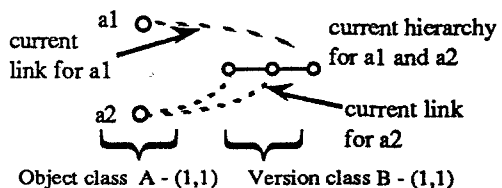
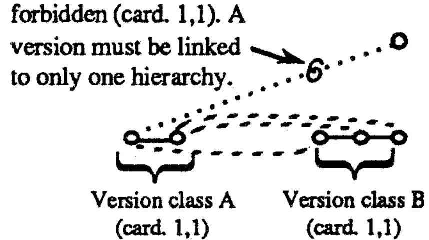


Fig. 5. Current hierarchies

If a relationship link is monovalued for the version class B, a version of B must be linked to a unique object of A. If the relationship link is multivalued for the version class B, a version of B must be linked to one or more objects of A.

Case 3 : relationship between version classes. This case expresses a link between an object version and one or more hierarchies of object versions.

If a relationship link is monovalued for the class A, a version of A is linked to only one hierarchy of B (and vice versa). If the relationship link is multivalued for the class A, a version of A must be linked to one or more hierarchies of B (and vice versa).



6.2 Operations

Operations performed on instances (create, delete, derive) must be studied in details. The other ones are available as described in section 3.6. The case 1 of relationship between object classes, widely studied in the literature (OMT, OOA, OOM, O*, ...), is not presented here.

Creation.

Case 2 : relationship between an object class and a version class. If the relationship link is monovalued for the object class A, a new object "a" created in A is linked to only one version "b" of the version class B. If the relationship link is multivalued for the class A, a new object "a" in the class A must be linked to one or more versions "b" of the class B. These versions belong to distinct hierarchies.

If the relationship is monovalued for the version class B, a new object created in the class A is linked to a version "b" of B which is either a leaf of a hierarchy of B not yet linked to an object belonging to A (if it exists), or versions derived from leaf versions of free hierarchies or a root of a new hierarchy of B. Multivaluation for the class B does not restrict the set of leaf versions which must be linked to an object "a".

If the relationship link is monovalued for the version class B, a new version "b" created in B (it is the root of a new hierarchy) is only linked to one object "a" of the object class A. If the relationship is multivalued for B, the new version "b" must be linked to one or more objects "a" of the class A.

Moreover, if the relationship link is monovalued for the object class A, the objects "a" which must be linked to the version "b" are either new objects of A which are not linked to a hierarchy of B (if they exist) or free objects.

A free object is an object which is not linked to a leaf of a hierarchy of B (cf 4.2.1).

If the relationship link is multivalued for the class A, an object of the class A can be linked to any version of the class B.

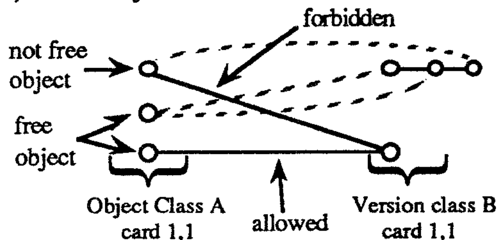


Fig. 6. Free objects

Case 3: relationship between version classes. If the relationship link is monovalued for the version class A, a new version "a" created in A (it is a root of a new hierarchy) is linked to a unique version "b" of a hierarchy of the version class B.
If the relationship link is multivalued for the class A, a new version "a" created in A must be linked to one or more versions, each belonging to distinct hierarchies of the class B.

When the relationship is monovalued for the class B, the versions "b" linked to the version "a" are either leaves belonging to existing hierarchies of B not yet linked to a version of A (if they exists), or versions derived from leaf versions of free hierarchies, or roots of new hierarchies of B. Multivaluation for the class B does not restrict the set of leaf versions "b" which can be linked to the version "a".

Deletion. An instance (object or version) deletion is permitted when it is linked to objects or to working versions. On the other hand, if the instance is linked to frozen versions, deleting it causes the deletion of linked frozen versions. The relationship links are obviously deleted.

Derivation. Derivation is an operation which can only be performed on instances of version classes. Only case 2 and case 3 are studied.

Case 2 : relationship between an object class and a version class. If the relationship link is monovalued for the version class B, a new version "b" derived from a version belonging to a hierarchy of B is linked to a unique object "a" of the object class A. When the link is multivalued, the new derived version must be linked to one or more objects of the class B.

The objects "a" which must be linked to the version "b" are either objects linked to the version from which "b" is derived, or free objects (cf § 5.2.2).

Case 3 : relaticnship between version classes. When the relationship link is monovalued for the version class A, a new version derived from a version belonging to a hierarchy of A is linked to a unique version "b" of a hierarchy of B. Multivaluation for A allows to link the version "a" to one or more versions "b", each belonging to distinct hierarchies of B.

The versions "b" which can be linked to the version "a" are:

- versions linked to the version from which "a" is derived,
- derived versions from versions linked to the version from which "a" is derived,
- leaf versions belonging to hierarchies of B (if the link is multivalued for B) or derived versions from leaf versions of free hierarchies of B (if the link is multivalued for B),
- roots of new hierarchies.

When the relationship link is monovalued for B, the versions "b" which are linked to the version "a" must not be linked to versions belonging to other hierarchies of A. A multivalued relationship for B does not restrict the set of version "b" which can be linked to the version "a".

7 Conclusion

This paper has presented a conceptual model intended for object-oriented database modeling. This model allows to describe, in a unified framework, objects, object versions and class versions. Three kinds of classes are used for such a modeling: object classes, version classes and versionable classes. Object class instances are objects of which one keeps only the last state. The instances of version classes are object versions where only the value evolve whereas the instances of versionable classes are object versions whose value and schema evolve (the different significant states are kept).

This paper investigates, in greater details, the outcomes of modeling links between these different kinds of classes on objects and object versions. The considered links are inheritance, composition and relationship links. Their cardinalities are also taken into account. Moreover, operations on objects and classes are also discussed in this study. Such a study has never been done for conceptual models. But it is partially approached for logical models (database models):

- Inheritance and relationship are not investigated.
- Composition is tackled in most of system managing object classes. Composition between object classes (case 1) and composition between version classes (case3) are studied but composition between object class and version class (case2) is never met.

  On the one hand, we can observe that the solution we propose in case 1 is the same than the one proposed in Orion [13], and, on the other hand, we can observe that the solution we propose in case 3 avoids to duplicate versions describing the same states of an object evolution (Presage), and avoids dynamic computing of derived versions (Orion [8]): these are directly linked with their component objects.

Such a study allows to model object, version and versionnable classes from the conceptual level. This study can be re-used to extend the data models of the current object-oriented database design methods (OMT [17], OOA [6], OOM [16], O* [5], ...) so that they integrate version modeling capabilities.

References

1. M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik. *The object-oriented database system manifesto.* 1st Int. Conf. on Deductive and Object-Oriented Databases, Kyoto (Japan), Dec. 1989.
2. D. S. Batori, W. Kim. *Modeling concepts for VLSI CAD objects.* ACM Transaction On Database Systems, Vol 10, n°3, 1985.
3. D. Beech, B. Mahbod. *Generalized version control in an object-oriented database.* 4th Int. Conf. on Data Engineering, Los Angeles (USA), 1988.
4. A. Bjornerstedt, C. Hulten. *Version control in an object-oriented architecture.* Object-oriented concepts, databases and applications, Edited by W. Kim, F. Lochovsky, Addisson-Wesley publishing company, 1989.

5. J. Brunet. *Modeling the world with semantic objects.* IFIP TC8 Int. Conf. on the Object-Oriented Approach in Information Systems, Québec, Oct. 1991.
6. P. Coad, Y. Yourdon. Object-oriented analysis. Yourdon Press publishing company, 1990.
7. H.T. Chou, W. Kim. *A unifying framework for version control in a CAD environment.* 12th Int. Conf. on Very Large Database, Kyoto (Japan), Aug. 1986.
8. H.T. Chou, W. Kim. *Versions and change notification in object-oriented database system.* 25th Int. Conf. on Design Automation, Anaheim, June 1988.
9. M.C. Fauvet. Définition et réalisation d'un modèle de versions d'objets. 5èmes Journées Bases de Données Avancées, Genève (Suisse), Sept. 1989.
10. W. Käfer, H. Schöning. *Mapping a version model to a complex object data model.* 8th Int. Conf. on Data Engineering, Tempe (USA), Feb. 1992.
11. R. Katz. *Toward a unified framework for version modeling in engineering databases.* ACM Computing Surveys, Vol 22, n°4, 1990.
12. W. Kim, H.T. Chou. *Versions of schema for object-oriented databases.* 14th Int. Conf. on Very Large Databases, Los Angeles (USA), Aug. 1988.
13. W. Kim. *Composite object revisited.* 14th ACM Int. Conf. on Managment of Data, Portland (USA), June 1989.
14. G.T. Nguyen, D. Rieu. *Schema evolution in object-oriented database systems.* Data and Knowledge Engineering, n°4, North-Holland publishing company, 1989.
15. S. Monk, I. Sommerville. *Schema evolution in object-oriented databases using class versionning.* ACM SIGMOD record, Vol 22, n°3, September 1993.
16. M. Rochfeld. *Les méthodes de conception orientées objet.* Conférence invitée, Congrès INFORSID, Clermont-Ferrand (France), May 1992.
17. M. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. *Object-oriented modeling and design.* Prentice-Hall publishing company, Englewood Cliffs, 1991.
18. G. Talens, C. Oussalah, M.F. Colinas. *Versions of simple and composite objects.* 19th Int. Conf. on Very Large Databases, Dublin (Ireland), Sept. 1993.
19. S. Zdonik. *Version management in an object-oriented database.* Lecture Notes in Computer Science n°244, June 1986.
20. R. Zicari. *A framework for schema updates in an object-oriented database system.* 7th Int. Conf. on Data Engineering, Kobe (Japan), April 1991.