ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

B Novikov, University of St Petersburg, Russia and J W Schmidt, University of Hamburg, Germany. (Eds)

Advances in Databases and Information Systems, Moscow 1996

Proceedings of the International Workshop on Advances in Databases and Information Systems (ADBIS '96). Moscow, 10-13 September 1996

Semantic Rules to Propagate Versions in Object-Oriented Databases

C. Urtado and C. Oussalah

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

Semantic Rules to Propagate Versions in Object-Oriented Databases

Christelle Urtado
LGI2P / EMA- EERIE
Nîmes, France

Chabane Ouassalah
LGI2P / EMA- EERIE
Nîmes, France

Abstract

Object versions are a key feature of object-oriented database evolution. The applications targeted by OODBs (such as CAD or CASE applications) need to manipulate complex objects, i.e. objects linked to other objects by dependence relations (composition, inheritance, association, etc.) and, consequently, complex object versions. This paper investigates the existing approaches in complex object version management through version propagation, i.e. the automatic creation or destruction of groups of linked object versions. The main issues of version propagation are underlined: limitation of the scope, version referencing and ambiguous propagation cases management. The paper then presents the authors' approach: a user-customizable multi-strategy version propagation model in which the propagation capabilities, called strategies, can be declaratively described by means of rules and associated with each relation, according to its semantics. A comprehensive example of composite class and instance version propagation illustrates the capabilities of the model.

1 Introduction and Problematics

Object versions are becoming widely used in modern database applications such as computer-aided design (CAD), computer-aided software engineering (CASE) and knowledge representation. The use of versions in database applications commonly leads to the distinction of two categories of object versions [5, 15] :

- **system versions** which are created and manipulated by the system. These versions are hidden from designers and users of applications and are principally used in OODBs to support transactions [1], redundant information and multi-user access to data.
- **user versions** (or apparent versions) which are created by designers and users of applications to experiment with several versions of a given object. These versions represent design alternatives of an object, the history of objects, different versions of a software component, etc. and are used to model object evolution [3, 4].

This paper will not describe the advantages of the concept of version which are illustrated by various version models such as Iris [2], Encore [17, 19], Lincks [10], Mosaic [11], Orion [7], Version Server [8], Présage [13, 18], etc. Rather, it will focus on the contribution of user versions to a more expressive semantic description of objects in a database application.

The objects represented in typical database applications (i.e. CAD or CASE applications) can be considered as complex in that they are linked to other objects by dependence relations such as composition, inheritance, association, etc. The manual management of versions of such objects becomes a tedious task. To help both designers and end-users of applications, this task is sometimes automated through version propagation mechanisms that propagate the creation and destruction operations from an object version to the objects in relation with it via dependence relations. This paper aims in presenting such an object version propagation model.

In order to allow uniform management of all types of complex objects, we consider the version propagation capabilities to be relation dependent [14]. For example, a given class will not propagate version creation to its sub-classes and

super-classes (inheritance relation) in the same way as to its composites and components (composition relation). Similarly, a given instance will not propagate version creation to its composites and components (composition relation) in the same way as to its associated instances (association relation).

The object version propagation model defined in this paper presents the following characteristics :

- propagation in complex objects along any type of dependence relation,
- propagation of two operations : object version creation and object version destruction,
- propagation adapted to each relation and to each application thanks to the possibility for the application designer to customize the semantics of propagation by means of the description of propagation rules,
- advanced control of version propagation : limitation of the scope of propagation to avoid useless version proliferation, management of reference propagation and of ambiguous propagation cases.

Thus, our model distinguishes itself from other existing models because it supports multiple propagation strategies. A given relation can therefore, depending on the needs of an application, behave differently towards version propagation. It is also user-customizable meaning the description of propagation mechanisms is not definitively frozen by the model and can be adapted to a particular use by an application designer.

The remainder of this paper is organized as follows. Section 2 introduces the main problems encountered in defining a version propagation model with the various approaches chosen by the existing models. Section 3 describes our solutions to the version propagation problems. It is here that we define the basis of a user-customizable multi-strategy version propagation model. Section 4 presents the application of our model to version propagation in the composition graph (i.e. to composite objects). We then conclude with perspectives for this work.

2 Existing approaches

Object version propagation concerns the automatic management of object version creation and destruction when objects are too complex to be dealt with manually. After the creation (*resp.* the destruction) of an object version O, it consists in the automatic creation (*resp.* automatic destruction) of versions of objects linked to O by one of the relations taken into account by the propagation model. The existing object version propagation models generally cope with three key problems for the control of version propagation : limitation of the scope of propagation, version referencing and management of ambiguous propagation cases. This section introduces the various solutions proposed by the most representative propagation models.

2.1 Limitation of the scope of propagation

The ability of an application designer to control the scope of propagation is an important characteristic of a complex object version model [8]. Indeed, it is not always advantageous, when a version of an object O is created (*resp.* destroyed) to propagate version creation (*resp.* version destruction) in the whole graph of the objects linked to O by dependence relations. The model must, in fact, automatically realize the creation (*resp.* the destruction) of versions while not inducing a proliferation of useless versions (*resp.* not destroying too many versions). The existing version propagation models can be classified into three groups, depending on the propagation technique employed. These propagation techniques determine how the existing models solve the problem of limitation of the scope of propagation :

- **Notification of change.** Used in Orion [6, 7] for simple referencing relations (i.e. relations with no predefined semantics, in opposition with, for example, the composition relation), it is an interactive mechanism in which the user has to validate the changes proposed by the system. It never automatically propagates operations on versions beyond the immediately linked objects.
- **Systematic propagation.** Used for example in Encore [16] for the inheritance relation and in Version Server [8] for the composition and equivalence relations, it consists in propagating recursively the creation of versions until the root or the leaves of the graph of the relation concerned are reached. It creates a very large number of object versions without proposing any means of controlling the scope of propagation to the designer.

- **Selective propagation.** It consists in propagating versions only through the relations that have been designated as being version propagation sensitive. It is the most current technique and is used, for example, in Encore [19], Mosaic [11] and Présage [13, 18] for the composition relation and in Iris [2] and Mosaic for the simple referencing relation. Since propagation takes place only on those relations that have been declared sensitive by an application designer, it stops in a branch of the graph when it encounters the first non-sensitive relation. Selective propagation therefore appears to be, in itself, a means of controlling the scope of propagation.

The limitation of the scope of propagation is an important characteristic of a version propagation model because it allows to avoid a proliferation of useless versions. Most of the models offer a solution through selective propagation. However, the control exerted on the scope of propagation is implicit. To our knowledge, no existing version model allows the limitation of the scope of propagation to be explicitly defined thus allowing the transitivity of the propagation to be stopped (*cf.* Section 3.2).

2.2 References

When two objects A and B are linked by a relation r, their respective versions may or may not be linked. If the versions are linked, the reference may then be static or dynamic.

Figure 1: Version referencing

We have identified four referencing possibilities between versions :

- 1) Versions of A and versions of B are not linked : A and B evolve separately (*cf.* Figure 1a). This situation can be useful to test A (*resp.* B) regardless of the relation linking it to B (*resp.* A). In this case, versions of A or B can be disconnected from the graph of the relation r, a situation which reduces the semantic expressivity of the object graph.
- 2) A copy of the relation r links a version of A to a version of B (*cf.* Figure 1b). The duplicated relation expresses the static link that exists between the versions of A and B but the duplicated relation is no longer semantically related to its origin relation r. This mechanism is the most frequently used for relation propagation ; for example, in Encore [17] for the inheritance relation and in Iris for the simple referencing relation. Mosaic, Orion and Présage also duplicate relations to link object versions.
- 3) A version of the relation r links a version of A to a version of B (*cf.* Figure 1c). This solution allows two object versions to be linked with a static reference while maintaining the semantic link between the relations. It also forces to create the relation version management mechanisms but allows the entites concerned (object versions and relation versions) to be dealt with homogeneously. This mechanism is used in Version Server [8] where the creation of a version of an extremity of either a composition or an equivalence relation triggers the creation of a version of the relation having the new object version as an extremity.
- 4) The object versions are linked by dynamic references (*cf.* Figure 1d). This approach allows the objects referenced by a given object to be dynamically changed. According to Katz [8], the possibility to reference objects dynamically is particularly useful during the design process and enables the designer to experiment with the quality of several alternatives. A drawback of this approach is that the current version (B' in Figure 1d) is unique. At any given time, all the versions of a same object A (here A' and A'') are considered to be dynamically linked to the same version of B (here B'). Moreover, if certain version combinations are inconsistent, there is no mechanism to guide the user in his/her choice of the current version. A dynamic reference based on the current version principle is equivalent to

declaring all version combinations consistent.

No existing model, to our knowledge, allows the application designer to choose the way in which references are going to be propagated by letting him/her choose from among the cases described above.

2.3 Management of ambiguous propagation cases

Propagation is said to be ambiguous when more than one path can lead from one object to another [8]. In Figure 2, there are two paths which go from D to A : D-B-A and D-C-A. The propagation of the creation of a version of D can generate four possible configurations, represented in Figure 2 :

- the creation of a version of B and A without creating a version of C (case 1),
- the creation of a version of C and A without creating a version of B (case 2),
- the creation of a version of B, C and A (case 3),
- the creation of a version of B and C and the creation of two versions of A (case 4).

Figure 2: A case of ambiguous propagation and the four possible configurations

When faced with cases of ambiguous propagation, a version propagation model must either propose a propagation strategy (i.e. choose a configuration from among the four) or provide the designer with tools enabling him/her to choose the configuration he/she wants to obtain. To our knowledge, only two version models provide the designer with such mechanisms : Mosaic [11] and Version Server [8]. For example, the group check-in operation of Version Server groups together in a single "transaction" the inclusion of several object versions in a composition graph. This approach aims to create only a single version of each object of the composition graph, whatever the number of composition paths linking two objects may be. Lincks [10], unlike Mosaic or Version Server, does not allow the designer to choose between cases 3 and 4 (*cf.* Figure 2). Instead, it always creates a single version of each composite object, even if it can be reached by several composition paths (case 3, Figure 2).

2.4 Drawbacks of existing models

2.4.1 Small number of relations taken into account

The existing version propagation models only take a fixed number of relations into account in their propagation mechanisms :

- the composition relation (Encore [19], Lincks [10], Mosaic [11], Version Server [8] and Présage [18]),
- the inheritance relation (Encore [16]),
- the equivalence relation (Version Server [8]),
- the simple referencing relation (Iris [2], Mosaic [11], Orion [7]).

To our knowledge, no existing model provides a generic version propagation mechanism allowing new relations to be taken into account for version propagation.

2.4.2 Fixed propagation strategies

For each managed relation, existing version propagation models only propose a fixed number of propagation strategies per operation. Some models (Encore for the inheritance relation, Orion for the simple referencing relation and Version Server for the composition and equivalence relations) propagate versions systematically along a relation. Such a propagation technique is equivalent to fixing a propagation strategy per relation.

Most of the version management models (Encore, Mosaic and Présage for the composition relation and Iris and Mosaic for the simple referencing relation) propagate version creation selectively. Among these models, the majority defines an imposed propagation direction. Only Présage authorizes propagation in both directions. These models therefore propose a maximum of two or four propagation strategies.

No model, to our knowledge, allows either the propagation strategy for a relation to be modified or new strategies to be created if needed.

2.4.3 Lack of distinction between class version propagation and instance version propagation

Few models distinguish class version propagation from instance version propagation by providing distinct propagation strategies. Encore [16] propagates version creation from classes to sub-classes, an operation which is not carried out on instances. Présage propagates class version creation or destruction differently than instance version creation or destruction. For example, a weak exclusive composition link propagates, as a default setting, instance version creation from composites to components whereas it does not propagate class version creation. Moreover, the distinction of class version propagation strategies from instance version propagation strategies allows the impact of class version creation on instances to be lessened. This subject is beyond the scope of this paper.

2.5 Conclusion

The chart of Figure 3 synthesizes the various solutions given by the most representative version models to the problems raised in this section.

3 A user-customizable multi-strategy version propagation model

The object version propagation model defined in this paper lies on the version model of Présage. It aims in automating the management of complex object versions. It therefore chooses to attach the propagation capabilities, called propagation strategies, directly to the relations involved [14].

3.1 Basic framework of the propagation model

3.1.1 General presentation

Our version propagation model is built upon two concepts (*cf.* Figure 4) :

- **Propagation rules.** They define the operations to be carried out for version propagation. For example, a propagation rule might indicate that the creation (*resp.* destruction) of a version of the extremity of a relation must trigger the creation of a new version (*resp.* the destruction of the version) of the relation as well as a version (*resp.* the destruction) of its other extremity. The propagation rules describe, in a declarative manner, the different propagation mechanisms that can be used.
- **Propagation strategies.** They describe what the behavior of a given relation should be regarding version propagation. This behavior is defined for two distinct operations : the creation of and the destruction of a version of one of the extremities of this relation. A propagation strategy is composed of one or more propagation rules for the creation of versions and one or more propagation rules for the destruction of versions.

Semantic Rules to Propagate Versions in Object-Oriented Databases

	Encore	Iris	Lincks	Mosaic	Orion	Version Server	Présage
Relations taken into account	- composition - inheritance	references	composition	- composition - references	references	- composition - equivalence	composition
Propagation technique	- systematic (inheritance) - selective (composition)	selective	selective	selective	change notification	systematic	selective
Implicit limitation of the scope	yes (composition)	yes	yes	yes	no	no	yes
Explicit limitation of the scope	no	no	no	no	limited to a single level	no	no
References	duplication (case 1b) for inheritance	duplication (case 1b)	- separated evolution for static links (case 1a) - dynamic reference propagation (case 1d)	duplication (case 1b)	duplication (case 1b)	relation version (case 1c)	duplication (case 1b)
Management of ambiguous cases	no	no	creation of a single version of the composite	pend-versions/ unpend-versions	no	group check-in / group check-out	no
Number of propagation strategies	- one for inheritance - two for composition	two	two	two for each relation	one	one for each relation	four
Distinction between class and instance version propagation	- yes for inheritance - no for composition	no	no	no (no class versions)	no	no (no class versions)	yes

Figure 3: Characteristics of the existing version propagation models

Figure 4: Basic framework of the version propagation model

A propagation strategy, grouping several propagation rules for creation propagation and several propagation rules for destruction propagation, is associated to each relation. This strategy, called the default propagation strategy for the relation, can either be directly associated with the relation or inherited from the super-classes of the relation.

The concepts of propagation rule and propagation strategy give the following characteristics to our model :

- the possibility to create new rules, thanks to their declarativity ; for example, by specializing existing rules,
- the possibility to change the default propagation strategy of a given relation,
- the possibility to create new strategies by combining existing rules,
- the possibility to create new strategies by combining newly defined rules,
- the possibility to create new strategies by specializing existing ones.

Figure 4 gives an overview of the objects used in the version propagation model framework.

3.1.2 *Predefined propagation rules*

The model possesses, by default, predefined rules that have been created to suit the needs of standard inter-object relations (such as composition, inheritance, etc.). The introduction of propagation rules allows access to a wide variety of version creation and destruction propagation techniques from among which the application designer can choose the one that best adapts to a given relation and a given application. Our version propagation model distinguishes version creation propagation from version destruction propagation. Each relation can propagate according to four propagation directions (FORWARD, BACKWARD, BIDIRECTIONNAL or NONE) and two propagation modes (RESTRICTED or EXTENDED) [14]. The semantics of these values are the following :

- When an operation propagated in the FORWARD direction is triggered on the source of a relation, it is propagated to the relation if the propagation mode is RESTRICTED (rule $\uparrow$) and to both the relation and the destination of the relation if the propagation mode is EXTENDED (rule $\uparrow\downarrow$).
- When an operation propagated in the BACKWARD direction is triggered on the destination of a relation, it is propagated to the relation if the propagation mode is RESTRICTED (rule $\downarrow$) and both to the relation and the source of the relation if the propagation mode is EXTENDED (rule $\downarrow\uparrow$).
- When an operation is associated with the propagation direction NONE (rule $\times$), it is not propagated via this relation.

An additional direction, BIDIRECTIONNAL, is introduced to model propagation in both FORWARD and BACKWARD directions. A pair of modes must be associated with this direction. (RESTRICTED, EXTENDED), for example, means that the propagation in the FORWARD direction is carried out with the RESTRICTED mode and that the propagation in the BACKWARD direction is done in the EXTENDED mode. This can be symbolized by the juxtaposition of two rules : $\uparrow\downarrow$.

The definition of a direction and mode of propagation allows the following problems to be solved :

- implicit limitation of the scope of propagation (by a technique which is equivalent to selectivity, as defined in Présage),
- version referencing : cases 1a and 1c of Figure 1 immediately result from an adequate choice for the direction and mode of propagation,
- management of ambiguous cases : cases 1 and 2 of Figure 2 also result from an adequate choice of the direction and mode of propagation.

3.1.3 Propagation strategies

The version propagation model we propose allows multiple propagation strategies to be expressed. A strategy is defined as a set of propagation rules (for creation and destruction) in which rules can be redefined or added. Furthermore, strategies themselves can be specialized to constitute a hierarchy of strategies. Our propagation model defines :

- **a default propagation strategy** associated with each type of relation. This strategy is provided as a default setting for each relation we have identified and can either be effectively used by the end-user or changed by the application designer for a strategy of his/her choice among the strategies proposed for this relation.
- **a set of admissible strategies** for each type of relation. This set, set by default or changed by the application designer, groups together strategies that are compatible with the semantics of the relation. It guides the designer of an application in his/her choice of a suitable propagation strategy. The modification of this set of strategies can be the source of an incompatibility with the semantics associated with the relation.

Our version propagation model further distinguishes between strategies for class version propagation and strategies for instance version propagation.

The concept of propagation strategy allows our model to solve two of the previously defined problems :

- by giving the application designer the possibility to choose, not only between a fixed number of propagation strategies but also between multiple strategies,
- by distinguishing between strategies for class version propagation and strategies for instance version propagation.

3.2 Explicit limitation of the scope of propagation

When a relation propagates the creation of object versions with the EXTENDED mode, it may be necessary to limit the effect of some modifications on some objects. The example in Figure 5 gives an illustration of this.

Let us suppose that all relations propagate version creation in the FORWARD direction and with the EXTENDED mode and that the designer wants the creation of a version of A not to propagate on C (*cf.* Figure 5b) but, on the contrary, wishes the creation of a version of B to propagate on C (*cf.* Figure 5c). This type of situation cannot be solved by simply using the direction and mode of the propagation. In fact, the relation between B and C should propagate from B to C with the RESTRICTED mode when the version creation is initiated by A and with the EXTENDED mode when the version creation is initiated by B.

To enable this type of situation to be expressed and, therefore, the transitivity of propagation to be explicitly stopped, we define an **inhibition link** which goes from an object S to a relation r. It indicates that when r, propagating with the EXTENDED mode, is crossed by the propagation process initiated by S, r should be considered as being in the RESTRICTED mode.

Figure 5: Example of explicit limitation of propagation

In our model, the setting of an inhibition link adds an activation condition to the propagation rule(s) concerned. Therefore, the scope of propagation can be controlled in two ways :

- implicitly with the direction and mode of propagation,
- explicitly with the high level inhibition mechanism.

This last point responds to the problem previously identified on the existing models.

3.3 References

Our model allows the four version referencing cases represented in Figure 1 to be obtained.

A separate evolution of the two extremities of a relation (case 1) is obtained, for example, by declaring the relation non-versionable.

Relation duplication (case 2) is possible in our model. The designer can manually position a relation between two separately evolving versions. In the context of automatic version management, such a manipulation can, in most cases, be replaced by relation version creation (which is often equivalent and automatic).

Relation version creation (case 3) can be obtained, for example, by propagating version creation with a BIDIRECTIONAL direction and the EXTENDED mode.

Our model is capable of dynamic referencing (case 4). A precise study of dynamic reference propagation (management of both the history of dynamic reference resolution and coherent configurations) is beyond the scope of this paper.

Unlike the capabilities provided by existing models, our version propagation model allows the type of version referencing needed to be specified.

3.4 Management of ambiguous propagation cases

In the case where an object can be reached by several paths in the graph of a relation, two solutions are commonly presented [11, 10, 8], generally to suit the composition graph. The most frequently used solution consists in creating a single version of each object of the graph, whatever the number of paths is. This approach allows to have a version graph isomorphic to the object graph. It is the default choice in our model. However, it is also possible, for the application designer, to disambiguate an ambiguous propagation case by choosing the case 4 presented in Figure 2. To do so, he/she can choose a propagation rule in which the attribute `MultiplePaths` is set to the value `multiple`. The default value is `unique` : it allows a single version of the destination object to be created, even if it can be reached by n different paths. When the value is set to `multiple`, n versions of the destination are created (one per path). The functionality provided by our model for the management of ambiguous propagation cases comparable to the solutions proposed by Mosaic and Version Server but offers the possibility to uniformly be extended to any type of relation. Ambiguous propagation cases can also exist in an object graph having cycles or circuits constituted by several different relations. To our knowledge, no existing version model takes this case into account. In our model, this type of ambiguity can be managed with the technique already used for the management of ambiguities where a single type of relation is involved. This can be done because :

- propagation strategies are associated to relations,
- we have not made any hypothesis on the structure of the graphs that our technique can handle.

Our model responds both to the ambiguous propagation cases presented in Section 2.3 and to the ambiguous propagation cases involving several types of relations each having its own semantics.

3.5 Conclusion

The model presented in this paper presents the following characteristics (*cf.* Figure 6) :

- propagation in complex objects along any type of dependence relation,
- propagation of two operations : object version creation and object version destruction,

Relations taken into account	Propagation technique	Implicit limitation of the scope of propagation	Explicit limitation of the scope of propagation	References	Management of ambiguous propagation cases	Number of propagation strategies	Distinction between class and instance version propagation
- composition and inheritance by default - extensible by the designer	selective	yes	inhibition link	four possible types of references	as the designer chooses	- strategies by default - extensible by the designer	yes

Figure 6: Characteristics of the version propagation model presented in this paper

- propagation adapted to each relation and to each application thanks to the possibility for the application designer to customize the semantics of propagation by means of the description of propagation rules,
- distinction between strategies to be used for class version propagation and strategies to be used for instance version propagation,
- distinction between default strategies and admissible strategies allowing the application designer to exert a certain control on the strategies that can be chosen for propagation via a given relation,
- implicit and explicit limitation of the scope of propagation to avoid useless version proliferation,
- management of reference propagation by letting the application designer choose the way in which references are propagated,
- management of ambiguous propagation cases in a way comparable to the one used in Mosaic and Version Server but with the possibility to be used homogeneously for any type of relation.

Thus, it distinguishes itself from other existing models because it supports **multiple propagation strategies**. A given relation can therefore, depending on the needs of an application, behave differently towards version propagation. It is also **user-customizable** meaning the description of propagation mechanisms is not definitively frozen by the model and can be adapted to a particular use by an application designer.

4 Application to composite object version propagation

Having presented a generic model for complex object version propagation, this section aims in showing a concrete application : composite object version propagation, i.e. propagation via the composition relation.

4.1 The composition relation

The choice to illustrate our propagation model with the composition relation lies in the fact that this particular relation is frequently used in object modelling. The composition relation has yet been the theme of many research works [9, 12] and its semantics is thus particularly well defined. The purpose of this paper is not to propose a new semantics for this relation but to consider the composition relation as it is usually defined and show how its characteristics can be modelled with our semantic rules to propagate versions in a composition graph.

Here is a brief description of the semantics of this relation. A composition relation can be :

- exclusive or shared,
- dependent or independent,

- predominant or non-predominant.

These characteristics are set at the class level but only apply on instances. The property of exclusivity / sharability indicates whether the referenced component can or cannot be shared by several composite objects. The properties of dependence / independence and predominance / non-predominance specify the existential dependencies of the objects. The existence of a component instance referenced by a dependent link depends on the existence of its composite while the existence of a composite instance referencing a component with a predominant link depends on the existence of this component.

4.2 Version propagation in the composition graph

The semantics of composition and, more particularly, its dependence and predominance properties, set the semantics of instance version destruction propagation. In the case of composite object version creation, we consider ourselves to be in a context of automatic object version management. We therefore consider the creation to be the symmetrical operation of the destruction.

In order to illustrate instance version propagation via composition links having different semantics, let us consider the example in Figure 7. A class *Car* is composed of a class *Engine* and a class *Body*. Let us suppose that the composition relations l_1 between the car and the body and l_2 between the car and the engine are dependent and predominant.

Figure 7: Example of a composite object

Figure 9 represents the propagation strategies and rules set by default for the relations l_1 and l_2 , both at the class and instance levels.

Let us suppose that the application designer decides to create a new version of the class *Engine* by adding the attribute *injection*. The propagation strategy S_1 ruling class version propagation for the relation l_2 implies the activation of the creation rule R_1 that creates a version of both the relation class l_2 and the class *Car* with the modification of the attribute *type* (this class attribute takes the value *GTI*). No version of the class *Body* is created (*cf.* strategy S_0 , rule R_0). The version schema obtained after this operation is represented on Figure 8.

Figure 8: Class versions after propagation

Let us now suppose that *my-car* is an instance of *Car* linked to *my-carbody* and *my-carengine* by two composition links *my-l₁* and *my-l₂* respectively instances of l_1 and l_2 (*cf.* Figure 10).

If the end-user decides to create a new instance version of *my-carbody* by changing the value of the attribute *color* (the car is painted in *metallic grey*), the system automatically creates (*cf.* strategies S_2 , rule R_2 and S_3 , rule R_3) a version of *my-l₁*, of *my-car* by changing the value of the attribute *quoted value* (a metallic paint increases the quoted

 Figure 9: Propagation strategies and rules associated to l_1 and l_2

Figure 10: An instance of Car

value of the car), of my- l_2 and of my-carengine. These changes are due to the dependent property of the relation l_1 and to the predominant property of the relation l_2 and are represented on Figure 11.

Figure 11: Instance versions after propagation

If the end user finally decides to destroy the version of my-carbody, the versions of the link my- l_1 and of my-car will be destroyed because of the predominance of l_1 (strategy S_2 , rule R_5). Consequently, and because of the dependence of l_2 , the versions of my- l_2 and my-carengine will be destroyed (strategy S_3 , rule R_6 , inherited by S_3 from S_2).

5 Conclusion and perspectives

In this paper, we have extended the version model of Présage in a user-customizable, multi-strategy version propagation model. This model may be distinguished from the existing models in that it associates the version propagation capabilities to the relations. Furthermore, it is generic in that it allows the application designer to customize the version propagation capabilities in defining propagation strategies. The model presented in this paper also responds to the commonly referenced problems of limitation of the scope of propagation, version referencing and ambiguous propagation case management. Moreover, we have applied this version propagation model to the management of propagation via the two most frequently used relations : the inheritance and composition relations. Future perspectives concerning this work include the definition of a taxonomy of propagation strategies and the study of a means to control the internal consistency of user-defined propagation rules.

References

- [1] Agrawal R, Jagadish H V. On correctly configuring versioned objects. In: proceedings of the 15th conference on Very Large Data Bases, Amsterdam, 1989, pp 367-374
- [2] Beech D, Mahbod B. Generalized version control in an object-oriented database. In: proceedings of the 4th IEEE International Conference on Data Engineering, Los Angeles, CA, February 1988, pp 14-22
- [3] Bounaas F. Management of evolution in knowledge base. PhD Thesis, Institut National Polytechnique de Grenoble, Grenoble, France, October 1995, (*in french*)
- [4] Casais E. Managing evolution in object-oriented environments : An algorithmic approach. PhD Thesis, University of Geneva, Switzerland, 1991
- [5] Cellary W, Jomier G. Apparent versioning and concurrency control in object-oriented databases. In: proceedings of the International Conference on Computing and Information, Peterborough, Canada, 1994
- [6] Chou H T, Kim W. A unifying framework for version control in a CAD environnement. In: Proceedings of the 12th conference on Very Large Data Bases, Kyoto, August 1986, pp 336-344

- [7] Chou H T, Kim W. Versions and change notification in an object-oriented database model. In: proceedings of the 25th ACM/IEEE Design Automaton Conference, Anaheim, June 1988, pp 275-281
- [8] Katz R H. Toward a unified framework for version modeling in engineering databases. ACM Computing surveys, December 1990, 22(4):375-408
- [9] Kim W, Bertino E, Garza J F. Composite objects revisited. In: proceedings of ACM SIGMOD, vol. 2, 1989
- [10] Lambrix P. Aspects of version management of composite objects. License thesis no. 328, Department of computer and information science, Linköping University, Sweden, 1992
- [11] Landis G S. Design evolution and history in an object-oriented CAD/CAM database. In: proceedings of the IEEE COMPCON conference, San Francisco, California, 1986, pp 297-303
- [12] Magnan M. Reusability of components : exceptions in composite objects. PhD Thesis, University of Montpellier II, Montpellier, France, September 1994, *(in french)*
- [13] Ouassalah C, Talens G, Colinas M F. Concepts and methods for version modeling. In: proceedings of the EURO-DAC'93 conference, Hamburg, Deutschland, September 1993
- [14] Rumbaugh J. Controlling propagation of operations using attributes on relations. In: proceedings of the OOPSLA'88 conference, September 1988, pp 285-296
- [15] Sciore E. Versioning and configuration management in an object-oriented data model. In: VLDB journal, 1994, 3(1):77-106
- [16] Skarra A H, Zdonik S B. The management of changing types in an object-oriented database. In: proceeding of the OOPSLA'86 conference, September 1986, pp 483-495
- [17] Skarra A H, Zdonik S B. Type evolution in an object-oriented database. In: Sower B and Wegner P (eds) Research direction in object-oriented programming, 1987, pp 393-415
- [18] Talens G, Ouassalah C, Colinas M F. Versions of simple and composite objects. In: proceedings of the 19th conference on Very Large Data Bases, Dublin, Ireland, August 1993
- [19] Zdonik S B. Object management system concepts. In: proceedings of the conference on Office Information Systems. ACM SIGOA, Toronto, Canada, June 1984, pp 13-19