Implementation of Composite Objects in an Object-Oriented Database System

Hafida BELBACHIR and Hiroshi ARISAWA***

* Institut d'Informatique. U.S.T.O. BP. 1505 EL M'NAOUAR, ORAN, ALGERIA
** Department of Electrical and Computer Engineering, Faculty of Engineering,
Yokohama National University. 156. Tokiwadai. hodogaya-ku, Yokohama 240,JAPAN

Keywords : object-oriented model. composite object. schema evolution.

Abstract

The composite object is a nested object on which the composite relationship is imposed. The concept of composite object is very important in many applications where the nested object is defined and manipulated as a single entity. However few object oriented systems take into account composite objects. In this paper, we propose an approach to modelize and to implement the composite objects. This approach extends the semantics of composite objects and reduces the time consuming when modification operations are performed on composite object.

Résumé

Dans les systèmes de base de données orientées objet, le domaine d'un attribut peut etre une classe. ce qui crée un lien de référence entre la classe à laquelle appartient l'attribut et la classe domaine. Cependant ce lien ne reflète aucune sémantique. Une relation spéciale dite relation de composition peut etre ajouter au lien de référence pour représenter le fait qu'un objet est une partie d'un autre objet dit objet composite.
Le concept d'objet composite est trés important pour plusieurs applications ou l'objet complexe est défini et manipulé comme une seule entité. Cependant trés peu de systèmes orientés objet prennent en compte cete notion.
Dans cet article, nous présentons une approche de modélisation et d'implémentation de l'objet composite. Cette approche augmente la sémantique du concept d'objet composite, et permet de réduire le temps consommé lors d'opération de manipulation des objets composites.

I. INTRODUCTION

In object-oriented systems, an object can have any number of attributes, and any of these attributes may take values from other classes (the values of the attributes are instances of other classes) [1][8][14]. This type of object is a nested object [2][3][6][9][11]. The nesting of objects does not capture any special relationship

between objects. Many applications [4][10][15], require the ability to define and manipulate a nested object as a single logical entity.

Also an important relationship may be superimposed on the nested object : the composite relationship [7][12][13][17]. The composite relationship captures the notion that an object is a part of another object. A composite object is that part of a conventional nested object on which the composite relationship is imposed. The composite object references its components through a composite attribute. This reference is called composite reference between the object and its component objects (also between the class of object and the class of components).

A composite object is defined as a directed acyclic graph where a node can be either a single component (with no component objects), or a root of a graph with its own component objects. A directed edge in the graph represents the composite relationship between a composite object and its component. This graph is called the composite graph.

The semantic of a composite reference is further refined by the concepts exclusive, shared, dependent, independent [13]. The composite reference may be of two types exclusive or shared. An exclusive composite reference from one object to another object means that the second object is a part of only the first object. Shared composite reference from one object to another object means that the second object is a part of the first object and possibly other objects.

A composite reference may be dependent or independent. A dependent composite reference from one object to another means that the existence of the second object depends on the existence of the first object, while an independent composite reference from one object to another means that the existence of the second object is independent of the existence of the first object.

The object oriented model of database management system ORION [12][13] is the first database system which supports the composite relationship (is-part-of relationship). ORION modelizes the composite relationship by maintaining in each component a list of reverse composite references, that is, object identifier of its composites.

A reverse composite reference consists of two flags in addition to the object identifier of the composite. The first flag indicates whether the object is a dependent component of the composite, while the second flag indicates whether the object is an exclusive component of the composite. However this approach suffers from a number of shortcomings. First, this representation is redundant since the type of the composite relationship between two classes C1 and C2 is represented in every instance of C2. Then this type is the same for every instance of C2, component of instances belonging to C1. In this paper, we propose to maintain the type of the composite reference only in the classes, and not in instances level.

The second shortcoming is the need to keep in each instance, the list of object identifiers of its composites. This approach is time consuming when one type of attribute is modified into another one also when schema change operations and instances operations are executed. To palliate to this second problem, an approach using the reverse pointers in a separate data structure is also proposed in this paper.

This paper is organized as follows. In section 2 we present briefly our object-oriented system which supports the composite relationship, then an example illustrating the composite reference types is given. The section 3 describes the implementation of composite objects. The section 4 formalizes the concept of composite reference, and gives the rules which must be verified by the types of composite reference. In section 5, we discuss the impacts of composite reference property on the semantic of some operations. The section 6 concludes this paper.

II. MODEL DEFINITION

The model [5] is based on a set of fundamental object-oriented concepts common to most object-oriented programming and knowledge representation (object identifier, attribute and method, class, inheritance). To take into account the application semantic the concepts of ``facet" and ``Integrity constraint" are added in the model. The facets [16] are attributes descriptors. They are used to modelize the sharing value, the inference (which permits to compute attribute value), the dynamic inheritance, the constraints (which can be defined on attributes to restrict their values) and the composite reference.

The class is defined by its name, its super-classes, its attributes, its methods and its integrity constraints (the integrity constraint is a restriction that need to be verified by the values of the attributes). An attribute is described by its name and a list of facets. We distinguish :

- *Type facets* (%one, %list-of, %set) : they specify if the attribute is single-valued (atomic), structured (its value is an object which belongs to another class called the attribute domain), or multi-valued (its value is a set of atomic values or a set of objects). The structured and multi-valued attributes create the attribute/domain link (or reference link).

- *Restriction facets* (%domain, %interval, %to-verify, %except, %mincard, %maxcard) : they limit the value of an attribute.

- *Value facets* (%const, %default, %init-value, %proc-attac) : they give or compute the value of an attribute.

- *Inheritance facet* (%inherited-from) which allows to solve the name conflicts arising from multiple inheritance. The user can specify the super-class from which the attribute is inherited.

- *Composite facets* (%composite, %exc, %dep) : the domain of an attribute of class C may be a class, and then creates an attribute/domain link between class C and domain class. This link can be composite reference link. Then the user can specify this composite reference by the facet %composite (value = true). As mentioned earlier, the composite reference may be shared or exclusive, independent or dependent. The user can specify the type of composite reference by using the facets %exc and %dep. The facet %exc specifies if the composite reference is exclusive or shared while, the facet %dep specifies if the composite reference is dependent or independent. The user can modify the type of composite reference. A detailed study is given in section 5.

III. EXAMPLE OF COMPOSITE REFERENCE TYPES

The following example (as shown in the figure 1), illustrates the types of composite reference. This example describes the database, called the ``neighborhoods management" which contains information about neighborhoods, their houses, gardens and schools. For the purpose of simplicity, we supposed that each neighborhood contains only individual houses, one school, one garden. The individual house is described by its rooms and its lot. The garden can be shared by the other neighborhoods but the school belongs to one neighborhood.

The class EDUCATION inventory all schools, universities, institutes,..

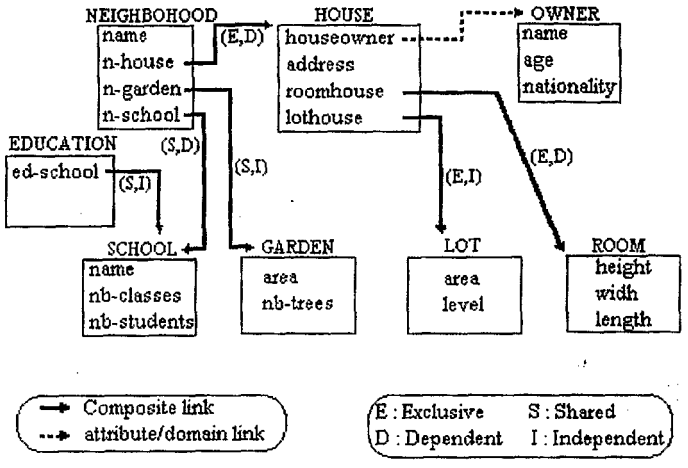

Figure 1 : Example of composite reference types

The house belongs to one neighborhood then the composite attribute *n-house* is exclusive dependent. Likewise a room is a part of only one house then the composite attribute *roomhouse* is exclusive dependent. The lot is a part of only one house but it can be used for another house when the first is destructed, then the composite attribute *lothouse* is exclusive independent.

A garden may be shared by many neighborhoods then the composite attribute *n-garden* is shared independent. The school is a part of only one neighborhood but it is inventoried by the education organization then the composite attribute *n-school* is shared dependent and the composite attribute *ed-school* is shared independent.

The attribute *houseowner* has domain class OWNER but it is not composite. The owner is not a component of house.

IV. COMPOSITE OBJECT IMPLEMENTATION

A class C references another class C', with composite reference, via a composite attribute A means that the value of the attribute A, of an instance x belonging to C, is the identifier of an instance y belonging to C'. So, every composite instance of C contains an instance identifier of class C'. With this representation, we can find, for each composite instance, the identifiers of its components directly or indirectly referenced from this instance via composite references.

Given a component, it is often necessary to determine its direct or indirect composites. Orion [13] proposes to maintain in each component a list of reverse composite references, that is, object identifiers of its direct composite objects. The number of reverse composite references in a component is equal to the number of its direct composites. A reverse composite reference consists of two flags in addition to the object identifier of a composite. One flag (D) indicates whether the object is a dependent component of the composite, while the other flag (X) indicates whether the object is an exclusive component of the composite. This approach is space and time consuming, since it requires to keep in every component the object identifier of the composite and two flags (D,X).

To prevent this disadvantage, the type of the composite relationship is maintained in the class rather than in every component, also the list of reverse composite references of each component is not kept in the instance component, but it maintained in a separate data structure. More precisely every component identifier with its composite identifiers are an instance of a pre-defined class INSTANCES.

The pre-defined class INSTANCES contains the identifiers of component instances with their composite identifiers. It is defined as following :

Defineclass INSTANCES
    attributes :
    # component : component identifier.
    # composite : list of composite identifiers.

This approach solves both the redundancy (the type of composite reference is maintained in the class rather in every component) and reduces the time consuming when modification operations are executed (instead of checking of and modifying the instances of the concerned class only some instances of pre-defined class INSTANCES are considered). The study of the semantic of these operations is detailed in section 5.

V. COMPOSITE REFERENCE CONSTRAINTS

An object may have many composite reference to it. These references may be of different types. However, some rules must be respected by these references. Theses rules are based on the semantic of composite reference types.

As said previously, the type of composite reference is represented in the classes, rather than in the objects, then we define the rules on the classes.

We first formalize the composite reference by a graph said class composite reference graph. Then we define the rules as conditions on this graph.

a) **Class composite reference graph**

Definition

A class composite reference graph G = (X, U) consists of a nodes set X and an edges set U, where the nodes set X is the classes set and the edges set U is defined as follows :

$$U = \{(c_1, c_2), c_1 \in X, c_2 \in X / c_1 \text{ has a composite reference to } c_2\}$$

This graph is further valued by the type of composite reference : every edge is valued by two values v1 and v2, v1 indicates whether the type of link is exclusive (v1 = 1) or shared (v1 = 0), while the other value indicates whether the type is dependent (v2 = 1) or independent (v2 = 0). This valuation is represented by a function V defined as follows :

$$D1 = \{1,0\}, D2 = \{1,0\}$$

$$V : U \rightarrow D1 \times D2$$

$$(c_1, c_2) \rightarrow V(c_1, c_2) = (v1, v2)$$

In figure 2, the class composite reference graph associated to the database neighborhoods management" is presented.

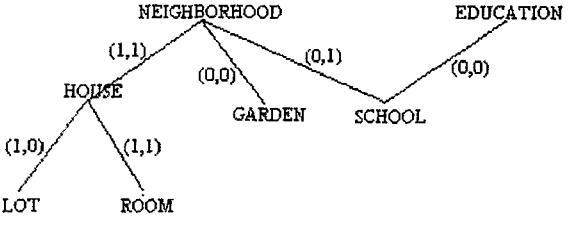

Figure 2 : Class composite reference graph

We denotes $\Gamma^+(c)$, the classes set which have a composite reference to c

$$\Gamma^+(c) = \{c_1 / (c_1, c) \in U\}$$

b) Graph conditions

condition 1
If a class c_1 has an exclusive composite reference to c, then c_1 is the only class which has composite reference to c. Formally

$$\forall c \in X(\exists c_1, c_1 \in \Gamma^+(c) \wedge V(c_1, c) = (1,-) \Rightarrow \Gamma^+(c) = \{c_1\})$$

condition 2
A class c has at the most one a dependent composite reference to it. Formally

$$\forall c \in X(\exists c_1, c_1 \in \Gamma^+(c) \wedge V(c_1, c) = (-,1) \Rightarrow \neg\exists c_2, c_2 \neq c_1 \wedge c_2 \in \Gamma^+(c) \wedge V(c_2, c) = (-,1))$$

These conditions on the graph are the rules which must be satisfied when modification operations are executed. We discuss now these operations and there semantics.

VI. OPERATIONS ON COMPOSITE OBJECTS

The operations on composite objects can be classified on three classes : 1) changes to composite reference type, 2) changes to the schema evolution, 3) changes to the instances. The rules defined previously must be preserved before and after any change operations.
In the following, let's suppose that a class C has a composite attribute A whose domain is a class D.

1) changes to composite reference type

Changes to the composite reference type include changing the composite reference to a non composite reference, changing an exclusive composite attribute to a shared composite attribute (and vice versa) and changing a dependent composite attribute to an independent (and vice versa).

1-a) *change a composite reference to a non composite reference*

This operation is executed without condition. The change from the composite attribute A to a non composite attribute is implemented by :

- changing the type of the attribute in the definition of class C (deleting the facets \%exc, \%dep and changing the value of the facet \%composite).
- accessing the instances of the class D in the pre-defined class INSTANCES and dropping reverse composite reference to instances of the class C.

1-b) *change an exclusive composite attribute to a shared composite attribute*

This operation is executed without condition. The change is implemented by modifying the value of the facet %exc in the class definition of C.

1-c) *change a shared composite attribute to an exclusive composite attribute*

The class D must not already have an exclusive composite reference to it (condition 1). The instances of the class D (in the pre-defined class INSTANCES) which are referenced through A, must have only one reverse composite reference (otherwise the operation is rejected). The change is realized by modifying the type in the class C (changing the value of the facet %exc).

1-d) *change a dependent composite attribute to an independent composite attribute*

This operation is executed without condition. The change is realized by changing the value of the facet %dep in the class definition of C.

1-e) *change an independent composite attribute to a dependent composite attribute*

If the composite reference is exclusive the change is realized without condition, but if it is shared, the class D must not already have a dependent reference to it (condition 2). The change is implemented by modifying the value of the facet %dep in the class definition of C.

2) schema evolution

Only schema change operations which involve a composite object are considered. These operations are : dropping a composite attribute or dropping a class.

2-a) *drop a composite attribute A in the class C*

If A is a dependent composite attribute, all instances that are referenced through A are deleted in the class D (deleting instances may imply deleting instances of another classes, see operation 3-b) and in the pre-defined class INSTANCES.

If the attribute A is independent, only reverse composite reference of instances belonging to class D to instance of class C are dropped, in the pre-defined class INSTANCES.

2-b) *drop an existing class C*

The instances belonging to the class C are deleted. If the class C has one or more dependent composite attributes, then all instances that are referenced through the dependent composite attribute by the instances of class C are deleted (deletion is recursively repeated).

If the class C was the domain of an attribute A1 of another class C1, A1 is assigned a new domain which is the first super-class of C.

All subclasses of class C become immediate subclasses of the super-classes of C.

3) changes to the instances

We discuss only about operations which have impact on the composite objects as, creating, deleting a composite instance, making an instance a part of composite object or deleting the composite reference between two instances.

3-a) *create a new composite instance x to a class C*

i) For each none composite attribute whose domain is a class then put the identifier of the referenced instance into x.

ii) For each composite attribute, if the referenced instance y exists and the composite reference is exclusive, then y must not already have any composite reference to it (otherwise the operation is rejected).

If the referenced instance y does not exist then creates the instance y and its associate instance in the pre-defined class INSTANCES.

Put the identifier of referenced instance y into instance x, and insert the identifier of x into the list of reverse composite reference of y (in the class INSTANCES).

3-b) *delete a composite instance x from the class C.*

i) If the instance x has a dependent composite reference to it then reject this operation else delete the identifier of x from the composite instance.

ii) Delete x from the class C and from the pre-defined class INSTANCES.

iii) For each composite attribute A that references a class D,

if the composite reference is dependent, then delete the referenced instances (deletion is recursively repeated) else delete the identifier x from the list of reverse composite references of y (into the class INSTANCES).

3-c) *delete a composite reference between instance x and instance y*

i) Delete the identifier y from the instance x.

ii) Delete the identifier x from the list of reverse composite references of y.

3-d) *make an instance y component of an instance x.*

i) If the composite reference is exclusive, then y must not already have any composite reference to it (otherwise reject the operation).

ii) Put the identifier of y into x, and insert the identifier of x into the list of reverse composites of y.

VII. CONCLUSION

The notion of composite object is very important in the new applications as computer-aided design, office automation, speech processing and so on, where the object must be treated as a whole. Also, to take into account the semantic of these applications, the notion of the composite reference is refined by the concepts of shared, exclusive, dependent and independent.

In this paper, we have presented an approach to modelize and implement the composite object in our object oriented database. The proposed approach permits to reduce the time consuming when modification operations are executed.

The semantics of these operations have been presented, showing the efficiency of our approach.

REFERENCES

[1] S.Abiteboul and S.Grumbach *``Bases de donnees et Objets Structures''*, TSI, Vol.6, No5, 1987.

[2] M.Adiba *``Modeling Complex Objects for Multimedia Databases''*, in Entity Relationship Approch : Ten Years of Experience in Information Modeling, S.Spaccapietra Ed., North Holland 1987.

[3] T.Andrews and C.Harris, *``Combined Language and Database Advances in an Object-Oriented Development Environment''*, In Proc.2nd Int.Conf.on Object-Oriented Programming Systems, Languages, and Applications, Orlando, Florida, Oct.1987.

[4] H.Afsarmanesh, D.Knapp, D.MCLeod and A.Parker, *``An Object-Oriented Approach to VLSI/CAD''* Proceedings of the International Conference on Very Large Data Bases, august 1985.

[5] L.Babahamed, *``Requetes de Consultation dans une Base de Donnees Orientes Objet''*, Thèse de Magister, Université d'Es-sénia, Algeria, 1993.

[6] J.Banerjee et al., *"Data Model Issues for Object-Oriented Applications"*, ACM Trans. on Office Information Systems, Vol5, No1, 1987.

[7] C.Djeraba and H.Brinand , *"A Design Object Concept"* , in Proceedings of International Symposium on Advanced Database Technologies and their Integration, Nara, Japan, october 1994.

[8] R.Ducournau , *"YAFOOL"*, Version 3.22 Manuel de Reference SEMA.METRA Montrouge, 1988.

[9] D.Fishman et al., *"IRIS : an Object-Oriented Database Management System"* , ACM Trans. on Office Information Systems, Vol.5, no1, 1987.

[10] S.Hudson and R.King , *"The Cactis Project : Database Support for Software Engineering"*, IEEE Transactions on Software Engineering, june 1988.

[11] W.Kim, H.T.Chou and J.Banerjee, *"Operations and Implementation of Complex Objects"*, in Proceedings of the Data Engineering Conference, Los Angelos, Calif., 1987.

[12] W.Kim, J.B.Banerjee, H.T.Chou, J.F.Garza and D.Woelk, *"Composite Object Support in an Object-Oriented Database System"*, in Proceedings of OOPSLA'87, 1987.

[13] W.Kim, E.Bertino and J.F.Garza, *"Composite Objects Revisted"* in Proceedings of SIGMOD, volume 2, 1989.

[14] R.Lorie and W.Plouffe, *"Complex Objects and their Use in Design Transaction"* in Proc.Databases for Engineering Applications, Database Week 1983 (ACM), San jose, California, May 1983.

[15] W.Kim and F.Lochovsky, *"Object-Oriented Concepts, Databases, and Applications"*, Addison-Wesley, 1989.

[16] F.Rechenmann, *"SHIRKA : Systeme de Gestion de Bases de Connaissances Centrees Objet"*, Manuel d'utilisation INRIA, Rocquencourt, 1987.

[17] M.E. Winston, R.Chaffin and D.A Herrmann, *"Taxonomy of Part-Whole Relations"*, Cognitive Science 11, 417-444, 1987.