Extending the ODMG Object Model with Composite Objects

Elisa Bertino
Dipartimento di Scienze dell'Informazione
Università di Milano
Via Comelico, 39/41 20135 Milano, Italy
+39 02 55006 227
bertino@dsi.unimi.it

Giovanna Guerrini
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova
Via Dodecaneso, 35 16146 Genova, Italy
+39 010 353 6635
guerrini@disi.unige.it

ABSTRACT

In this paper we extend the ODMG object data model with composite objects. A composite object is an object built by aggregating other component objects. Exclusiveness and dependency constraints, as well as referential integrity, can be associated with composition relationships among objects. Our composite object model is developed in the framework of the ODMG object database standard data model, but can be used in both object-oriented and object-relational database systems. In the paper, we propose a language for defining composite objects and we define the semantics of update operations on composite objects.

Keywords

Object-oriented database systems, composite objects, integrity constraints, data models.

INTRODUCTION

Object-oriented DBMS (OODBMS) and object-relational DBMS (ORDBMS) are establishing themselves as the new generation DBMS. Object database systems overcome the limitations of relational systems with respect to several emerging data-intensive applications because of their ability to directly represent complex objects and to store in the database not only data but also the operations that can be performed on the data.

All the data models supported by object DBMS share a number of basic concepts, such as the concepts of object, class, and inheritance, which are often collectively referred to as *core data model*. That core model, although quite rich, does not capture integrity constraints and semantic relationships which are important in many applications. In particular, an important semantic relationship is the *composition* relationship. In an object-oriented data model, an object, called *composite object*, can be defined by aggregating other objects, called *component objects* [19]. The relationship relating a composite object with its components is called *aggregation* or *part-of* relationship. The fact that an object is defined as an aggregate of other objects raises a number of requirements. A first relevant constraint is related to referential integrity. Referential integrity ensures that whenever the value of an attribute of an object is another object, the referenced object exists. Violations to referential integrity must be detected upon any attempt to delete a referenced object. Violations to referential integrity can be repaired by forbidding the deletion, by setting to null the value of the attribute containing the reference or by cascade deleting also the object containing the reference. In addition to referential integrity, two other important constraints are related to composite objects, namely exclusiveness and dependency. An exclusiveness constraint specifies that an object must be component of only one composite object (possibly, with respect to a certain class or through a certain attribute). A dependency constraint specifies whether the existence of the component object is independent from the existence of the composite objects containing it, that is, whether the deletion of a composite object implies the deletion of its components. In addition of being relevant from the semantic point of view, the notion of composite object as a logical entity is useful in enhancing performance because a composite object can be considered a locking, authorization and clustering unit.

In this paper we define a composite object model, supporting also referential integrity. A lot of work has been carried on composite objects and part-whole relationships, in the data modeling [3, 14, 15], knowledge representation [23, 24, 25] and software engineering [12, 20] areas. There is obviously a trade-off between the complexity of the model and the efficiency and simplicity of its realization. We restrict ourselves to composite object models typical of OODBMS. The model we propose unifies composite object models typical of OODBMS and referential integrity as supported in SQL. In particular, composite object models like the Orion one [19] do not include referential integrity constraints. Therefore, an object may be removed even if it is a component of a composite object. Referential integrity through the notion of external keys is supported in relational DBMS [8]. However, the notion of composite objects is not supported by SQL nor by the object extensions of SQL [11, 17, 22]; therefore, exclusiveness and existence constraints are not supported by these data models. Our goal is to define a composite object model supporting: (i) referential integrity with the same repair actions supported by SQL; (ii) exclusiveness constraints and existence constraints in a more general form than those defined in Orion [19].

We cast the definition of our composite object model in the framework of the ODMG object database standard data model [9]. However, we believe that our model, being quite general, can be used in both OODBMS and ORDBMS. The ODMG data model, which we describe in the following section, supports two kinds of properties of objects: attributes and relationships. Relationships are declared between two object types and induce a pair of traversal paths between the two types. In this paper, we extend the ODMG data model with a special kind of relationship, the *part-of* relationship, modeling composite object references. Note that the part-of relationship is listed among the features that will have to be addressed by the ODMG standard. Therefore, we believe that our proposal addresses an issue still open in the ODMG standard.

In particular, in this paper we propose a language, as extension of the ODMG object definition language (ODL), for defining composite objects. The language supports the definition of two kinds of relationships between objects, composite and weak relationships. Composite relationships have exclusiveness and dependence constraints. Exclusiveness is expressed by specifying a set of classes with respect to which exclusiveness is enforced. Moreover, for each relationship, the behavior to be observed upon violations of referential integrity can be specified. We then analyze update operations; in particular, the delete operation must be carefully handled. The *dependence* semantics of a composite reference as well as the the *cascade* option for referential integrity propagate the deletion of an object to other objects, while the *restrict* option for referential integrity prevents the deletion of referred objects. These different options may, thus, raise conflicts. In our work, we have developed strategies for solving ambiguities when conflicts arise. The basic idea behind conflict resolution is to adopt a *conservative* approach, that is, we prefer not deleting an object, which is no longer useful, rather than deleting a useful object. We remark that those conflicts depend on the dynamic configuration of the object aggregation graph, and thus they cannot be detected at the schema level. At the schema level, we could impose some sufficient conditions to prevent conflicts, but these conditions would be too restrictive. We exemplify and analyze those conflicts, and formally define a semantics of update operations.

The remainder of this paper is organized as follows. We first introduce some preliminary notions, then we present the language for defining composite objects and discuss the semantics of update operations on composite objects. Finally, we conclude the paper by discussing some implementation issues and pointing out some possible extensions.

PRELIMINARIES

In this section we introduce some preliminary notions that will be useful in the subsequent development of the paper. In particular, we introduce the notions of composite object and referential integrity, the basic concepts of the ODMG data model and all the notations that will be used in the following.

Composite Objects and Referential Integrity

Composite objects have been introduced in the Orion data model [18, 19]. In that model, two types of references - weak and composite - are defined between objects. A weak reference is a usual reference between objects on which no additional semantics is superimposed. An object o has a reference to an object o' if this reference is the value of an attribute of o. A composite reference is a reference on which the part-of relationship is superimposed. A composite reference can, in turn, be *exclusive* or *shared*. In the former case, the referred object must belong to a single composite object, whereas in the latter case it can belong to several composite objects. The semantics of a composite reference is then refined by introducing the distinction between *dependent* and *independent* composite references. In the former case, the existence of the objet referred to is dependent upon the existence of the object to which it belongs, whereas in the latter case, it is independent. The deletion of a composite object results in the deletion only of the component objects which are dependent for their existence. The objects whose existence is independent are not deleted. Obviously, because the dependence/independence is orthogonal with respect to the exclusiveness/shared status, four possible types of composite references are obtained. In case of shared dependent composite references, an object can be dependent upon several objects; this means that the deletion of a composite object results in the deletion of a shared component object only if all the other references to the object have been removed.

That model has been extended in [5] by introducing two new forms of exclusiveness for composite references: *exclusiveness with respect to a class*, meaning that two instances of the same class cannot share a component, whereas instances of other classes can refer that component; *exclusiveness with respect to a class hierarchy*, meaning that two members[1] of the same class cannot share a component.

Referential integrity defines a relationship among semantically identical attributes in different entities. In an object-oriented data model this means that for any object o containing a reference to an object o', the referred object, i.e. o', must exist. In some OODBMS, such as GemStone [7] and O_2 [13], referential integrity is automatically ensured, in that those systems do not support explicit object deletion, rather they use a *garbage collection* mechanism to determine which objects can be removed not being referred any longer by any other object. In systems with an explicit delete operation, if a referred object is deleted, the problem of *dangling pointers* may arise.

Referential integrity is not a peculiar constraint of object-oriented data models, rather it is present in all data models. The concept of referential integrity, is included in the SQL2 standard [8] which was accepted by ANSI and ISO in 1992. In SQL2 various options for referential integrity can be specified. Referential integrity in the relational data model is an integrity constraint between a set of attributes (called *foreign key*) of a relation C (called *child*) and a set of attributes that is the primary key of a relation P (called *parent*). Integrity constraint violations are caused by update operations on the parent and child relations. According to the SQL2 standard, insert and update operations on the child relation are forbidden (backed out) if these would result in database states violating referential integrity. For deletions and updates of tuples in the parent relation, by contrast, in SQL2 different repairing actions can be specified. This is accomplished through clauses ON UPDATE and ON DELETE in the definition of the parent relation, stating how to handle modifications that may violate referential integrity. For each of the clauses above, different repairing actions can be specified:

- CASCADE: in case of update, the new values in the key are propagated to the referencing children, whereas in case of deletion the referencing children are also deleted;
- SET_NULL: the foreign key attributes in the referencing tuples of the child relation are set to null;
- SET_DEFAULT: the foreign key attributes in the referencing tuples of the child relation are set to a given default value;
- NO_ACTION: no action is taken; referential integrity remains violated and if no other operation is executed to correct the mismatch of the corresponding tuples, the complete work of the transaction is backed out.

There is another important referential action not introduced in the SQL2 standard, but supported by most of the relational DBMS: RESTRICT. The semantics of this referential action is to forbid any change (update or delete) of a parent tuple as long as there are referencing child tuples. Although this action is not in the SQL2 standard, we include it in our discussion.

A problem with referential integrity constraints, as specified in SQL2, results from the possibility of interference when performing multiple referential actions on a tuple. That is, a straightforward implementation may lead to a *non-determinism* in the result of a user operation. The SQL2 standard prevents such non-determinism through the specification of a complex test carried out during the execution of referential actions. A detailed discussion of this approach can be found in the work by Markowitz [21].

ODMG Object Database Standard Data Model

This section focuses on the features of the object model of the ODMG standard that are relevant to this paper. The basic modeling primitives are the *object* and the *literal*. Each object has a unique identifier. A literal has no identifier. The state of an object is defined by the values it carries for a set of *properties*. These properties can be *attributes* of the object itself or *relationships* between the object and one or more other objects. Typically the values of an object properties can change over time. The behavior of an object is specified by the set of *operations* that can be executed on or by the object. Objects and literals can be categorized according to their *types*. All elements of a given type have a common range of states (i.e., the same set of properties) and common behavior (i.e., the same set of defined operations). A

[1] We say that an object is an *instance* of a class C if C is the most specific class, in the class hierarchy, to which the object belong. An object is a *member* of a class C if it is an instance of C or of a subclass of C.

Symbol	Meaning
OBJ	the set of all objects
$\mathcal{P}(S)$	the powerset of set S
$C_1 \leq_{ISA} C_2$	class C_1 is a subclass of class C_2
$\llbracket C \rrbracket$	the set of objects members of class C
$\llbracket C \rrbracket^+$	the set of objects instances of class C
$\llbracket S \rrbracket^+$	the set of objects instances of classes in the set S
$ref(o)$	the set of objects participating in some relationships with o
$comp(o)$	the set of objects participating in some composite relationships with o (the set of o components)
$in\_paths(o)$	the set of traversal paths leading to object o
$r(o)$	the object participating in relationship r with object o
r^{-1}	the inverse traversal path of traversal path r
$o \xrightarrow{r} o'$	there is a traversal path r from object o to object o'
$composite(r)$	whether or not relationship r is composite
$exclusive(r)$	the set of classes w.r.t. which relationship r is exclusive
$in\_rel(r)$	whether or not relationship r is exclusive in r
$dependent(r)$	whether or not relationship r is dependent
$delete(r)$	behavior to be taken upon deletion of an object from which traversal path r originates
$Set^d(o)$	the set of objects to be deleted as a consequence of the deletion of o

Table 1: Notation and terminology

database stores objects, enabling them to be shared by multiple users and applications. A database is based on a *schema*, defined according to an object definition language (ODL), and contains instances of the types defined by its schema.

A type defines a set of properties (attributes and relationships) constituting the state of instances of the type. An attribute models a property of all instances of a type. A relationship is defined between two types, each of which must have instances that can be referenced by object identifiers. Only binary relationships, i.e., relationships between two types, are supported. A binary relationship may be one-to-one, one-to-many, or many-to-many, depending on how many instances of each type participate in the relationship. For example marriage is a one-to-one relationship between two instances of type Person. A woman can have a one-to-many mother of relationship with many children. Teachers and students typically participate in many-to-many relationships.

A relationship is implicitly defined by declaring *traversal paths* that enable applications to use the logical connections between the objects participating in the relationship. Traversal paths can be declared in pairs, one for each direction of traversal of the binary relationship. For example, a professor teaches courses and a course is taught by a professor. The teaches traversal path would be defined in the declaration for the Professor type. The is_taught_by traversal path would be defined in the declaration for the Course type. The fact that both these traversal paths apply to the same relationship is indicated by an inverse clause in both the traversal path declarations. For example:

```
class Professor {   ...
  relationship Set<Course> teaches
      inverse Course::is_taught_by;
  ... }

class Course {  ...
  relationship Professor is_taught_by
  inverse Professor::teaches;
  ... }
```

However, the specification of an inverse traversal path is not mandatory; in such a case, the relationship can be traversed in a single direction. The relationship defined by the teaches and the is_taught_by traversal paths is a one-to-many relationship between Professor and Course objects. The cardinality is shown in the traversal path declarations. A Professor instance is associated with a set of Course instances via the teaches traversal path. A Course instance is associated with a single Professor instance via the taught_by traversal path.

The OODBMS is responsible for maintaining the referential integrity of relationships. This means that if an object that participates in a relationship is deleted, then any traversal path to that object must also be deleted. For example if a particular Course instance is deleted, then not only is the reference from this object to a Professor instance via the is_taught_by traversal path deleted, but also all references in Professor objects to the Course instance via the teaches traversal path must be deleted. Referential integrity ensures that applications cannot dereference traversal paths that lead to non-existing objects.

Notations

Table 1 illustrates the symbols most frequently used in this paper. For each symbol, the table reports a brief explanation of its meaning.

A LANGUAGE FOR SPECIFYING COMPOSITE OBJECTS

For the sake of simplicity in the presentation we restrict ourselves to consider single-valued, that is, one-to-one relationships. The extension to multi-valued (that is, one-to-many and many-to-many) relationships is straightforward.

In what follows we extend ODMG relationships to support the composite semantics for relationships. The actions to execute upon referential integrity violations can be specified for composite as well as for weak relationships. As we have discussed in the previous section, ODMG already supports a form of referential integrity, by deleting a relationship upon deletion of the participating objects. We extend that model by allowing a user to declare different behaviors, corresponding to the SQL CASCADE and RESTRICT options. Therefore, we can prevent the deletion of an object participating to a relationship and we can propagate the deletion of an object to the object which participates in the relationship with the deleted object. We can thus model mandatory relationships, that cannot be modeled by ODMG.

We extend the ODL syntax for specifying relationships as specified in Figure 1, according to a BNF-like style. In the grammar of Figure 1:

- the keyword composite specifies the relationship to be a composite relationship;
- the keyword exclusive introduces the set of classes with respect to which the relationship is exclusive; if the option in rel is specified, then the exclusiveness is intended with respect to the relationship itself;
- the keyword dependent specifies that, upon the deletion of the composite object, the component objects are also deleted, unless they participate in some other composite relationship;
- the keyword delete specifies the actions to be performed on the composite object upon the deletion of one of its components; this clause can be present also for weak references.

```
<rel_dcl> :: = [<compl_decl>]
                relationship <class_id> <rel_id>
                [inverse < inv_trav_path >]
                [delete < del_opt > ]

<compl_decl> ::= composite [dependent]
                [exclusive <excl_target>]

<excl_target> ::= [in rel] < target_classes >

< target_classes > ::= all | { <t_class> }

<t_class> ::= <class_id> | hierarchy(<class_id>) |
                <t_class>, <t_class>

<del_opt> ::= cascade | restrict

<inv_trav_path> ::= <class_id> :: <rel_id>
```

Figure 1: ODL extension for specifying composite relationships

We remark that it is not mandatory to specify an inverse relationship. If an inverse relationship is not specified the relationship is unidirectional, that is, it can be traversed in a single direction. If an inverse relationship is specified, certain coherence conditions must be verified by a relationship and its inverse relationship.

In what follows we discuss in detail the various clauses in the relationship definition.

Composite clause

If the keyword composite appears in a relationship definition, the relationship is qualified as a composite relationship. The semantic meaning of a *consists-of* relationship is thus imposed on the relationship. If the keyword is missing, the relationship is weak, that is, no special semantics is superimposed on it. Keywords exclusive and dependent can appear only in composite relationships. Note that the keyword composite must appear in the declaration of the traversal path associated with the composite object, and not in the traversal path associated with the component object.

Exclusive clause

The exclusive clause introduces:

- the option in rel, specifying that exclusiveness is restricted to the relationship itself;
- a set of classes, corresponding to the < target_classes > non-terminal of the grammar, with

respect to which the relationship is exclusive; in particular:

- the keyword all specifies the relationship to be exclusive with respect to all the classes in the database;
- the notation C, where C is a class identifier, specifies the relationship to be exclusive with respect to the instances of class C, that is, the relationship is exclusive with respect to class C;
- the notation hierarchy(C), where C is a class identifier, specifies the relationship to be exclusive with respect to the members of class C, that is, the relationship is exclusive with respect to the class hierarchy rooted at C.

The keyword all has been introduced to denote the set of all classes without the need of explicitly enumerating them. This option is useful in case of addition of new classes to the database.

If the clause is omitted, no exclusiveness semantics is associated with the relationship (that is, it is shared). A composite relationship can then be associated with a set S of classes. The relationship is thus exclusive with respect to all instances of classes in S. Set S is the empty set $\emptyset$ for relationships for which no exclusive clause is specified; it is the entire set of classes of the database for relationships for which exclusive all has been specified. Finally, S is defined as $\bigcup_{i=1}^{n} C_i \cup \bigcup_{j=1}^{m} \bigcup_{C^* \leq_{ISA} C'_j} C^*$ if the exclusive clause of the relationship definition contains $\{\mathsf{C_1}, \ldots, \mathsf{C_n}, \mathsf{hierarchy}(\mathsf{C'_1}), \ldots, \mathsf{hierarchy}(\mathsf{C'_m})\}$ with $m, n \geq 0$.

The following definition formalizes the notion of exclusiveness with respect to a set of classes S.

Definition 1 *(Exclusivity). Let r be a relationship specified in class C to be composite exclusive with respect to a set of classes S. The following conditions must hold:*

- $\forall o \in [\![C]\!] \quad \nexists\ o' \in [\![S]\!]^+,\ o \neq o'$, *such that* $r(o) \in comp(o')$;
- *if* $C \in S$, *moreover,* $\forall o \in [\![C]\!] \quad \nexists\ r'$ *composite relationship of class* C, $r' \neq r$, *such that* $r(o) = r'(o)$.

□

The first condition ensures that the object associated with o by r does not participate in any composite relationship with any other object instance of classes in S, while the second one ensures that that object is not associated with o in any other composite relationship.

The following definition formalizes the notion of exclusiveness in a relationship with respect to a set of classes S.

Definition 2 *(Exclusivity in a Relationship). Let r be a relationship specified in class C to be composite exclusive in r with respect to a set of classes S. The following condition must hold:*

$\forall o \in [\![C]\!] \quad \nexists\ o' \in [\![S]\!]^+, o \neq o'$, *such that* $r(o) = r(o')$.

□

Obviously, for the above notion to be meaningful, each class in S must participate to an r relationship.

Intuitively, exclusiveness with respect to a set of classes S states that two objects instances of classes in S cannot share a component, while exclusiveness in a relationship with respect to a set of classes S states that two objects instances of classes in S cannot be in the specified relationship with the same component. We remark that an exclusive relationship r is also exclusive in r, since $r(o') \in comp(o')$. Thus, exclusivity is a stronger notion than exclusivity in a relationship.

Dependent Clause

The dependent clause specifies that the deletion of the composite object must be propagated to the component objects. This automatic propagation is useful because it saves the application from having to explicitly delete all the component objects.

Consider a relationship r, specified in a class C to be composite dependent. Consider moreover an object $o \in [\![C]\!]$, such that $r(o) = o'$. Because the relationship is dependent, upon the deletion of o, o' is also deleted, if it does not participate in other relationships. The deletion of o', may in turn cause the deletion of other objects, if o' has some components. If the relationship is not dependent (that is, no dependent clause is specified), then the deletion of o does not impact o'.

Delete Clause

The delete clause specifies how the deletion of an object affects other objects eventually in relationships with this object. Consider a relationship r, specified in a class C. Consider moreover an object $o \in [\![C]\!]$, such that $r(o) = o'$. Then, the deletion of o' has different results depending on the value of the delete clause of r:

- if this clause contains the keyword cascade the deletion of o' implies the deletion of o;
- if this clause contains the keyword restrict o' cannot be deleted;
- if this clause is omitted the deletion of o' only results in the deletion of the traversal paths r and r^{-1} as in the current semantics of ODMG[2].

[2] Note that traversal paths r and r^{-1} are obviously deleted also in the cascade case.

Examples

In what follows we present two examples to illustrate the notions introduced above.

Example 1 *Consider the following class definitions.*

```
class Team {
  attribute string name;
  attribute short score;
  composite exclusive {hierarchy(Team)}
    relationship Player goalkeeper
    delete restrict;
  ...
  composite exclusive {hierarchy(Team)}
    relationship Player forward;
  composite exclusive {hierarchy(Team)}
    relationship Coach coach
    inverse Coach::trains
    delete restrict;
  composite dependent
    relationship Sponsor sponsor; }

class Coach {
  attribute string name;
  relationship Team trains
    inverse Team::coach; }
```

A team consists of a number of players, playing in different roles, and a coach. In particular, each team has a goalkeeper and may have a forward. Both relationships goalkeeper *and* forward *are composite, and exclusive with respect to the class hierarchy rooted at class* Team*. This models the fact that a player cannot play in two different teams, nor can play two different roles in the same team. This also means that two subclasses of class* Team*, say classes* First_Class_Team *and* Second_Class_Team*, cannot share a player. A player cannot be deleted as long as he is the goalkeeper of some team and a coach cannot be deleted as long as he trains a team. This models the fact that both* goalkeeper *and* coach *relationships are mandatory. A team consists moreover of a sponsor. Sponsors are dependent for existence on the existence of the team they sponsor. Finally, note that only relationship* coach *has an explicit inverse relationship* trains*.* ◇

Example 2 *Consider the following class definitions.*

```
class Project {
  attribute string name;
  attribute short budget;
  composite exclusive in rel {hierarchy(Project)}
    relationship Manager leader
    inverse Manager::leads
    delete cascade;
  composite dependent
    relationship Employee staff_member
    inverse Employee::works_in; }

class Employee {
  attribute string name;
  attribute short salary;
  relationship Project works_in
      inverse Project::staff_member; }

class Manager extends Employee {
  relationship Project leads
      inverse Project::leader; }
```

A project is characterized by a leader and a staff member. The same manager cannot lead two different projects, however, a manager (which is also an employee) can be the leader and the staff member of a project (or of different projects), because of the in rel *option in the exclusiveness declaration for relationship* leader*. The deletion of a project does not impact the existence of its leader; by contrast, upon deletion of a manager the project he eventually leads is in turn deleted. By contrast, an employee can be a staff member of several projects. The deletion of a project also results in the deletion of its staff member, if the member is not the staff member, or the leader, of other projects. If an employee is deleted, finally, the project he works for is not affected.* ◇

Correctness of Composite Relationships Definitions

It is important to note that given a traversal path declaration in a class, an inverse traversal path can be specified in the other class participating in the relationship. Therefore, some constraints must be imposed to ensure the coherence between a traversal path declaration and the declaration of the inverse traversal path. In particular, we choose to state that a relationship is composite in the declaration of the traversal path from the composite object to the component. Thus, if r is declared composite, r^{-1} must not be declared composite. Moreover, because a composite traversal path declaration specifies both the actions to perform upon the deletion of the composite object as well as upon the deletion of the component, no delete clause can be specified for the inverse of a composite traversal path.

Finally, we remark that specifying a dependent clause in a composite traversal path is different from specifying delete cascade in the definition of the inverse traversal path. The specification of the dependent clause requires the deletion of the component object upon deletion of the composite object, unless the component object is referred by some other object. By contrast, the delete cascade implies the deletion of the component object independently from other references to it (unless if they are through relationships with delete restrict, as we will discuss in detail in the following section).

SEMANTICS

In this section we consider the semantics of update operations on composite objects. We first consider creation and modification, and then deletion.

Creation and Modification

The creation of a new object may cause the violation of composite exclusive relationships already in the database. The creation is thus allowed if for any composite relationship in which the object participates, exclusiveness is not violated. Moreover, the creation may violate referential integrity if the object specified as value for a relationship does not exist in the database. If any of these constraints is violated, the creation is disallowed, that is, it is backed out. The following rule states the conditions upon which creation is allowed.

Rule 1 *(Creation Rule). The creation of object o is allowed provided that:*

- REFERENTIAL INTEGRITY*: for each object $o' \in ref(o)$, $o' \in OBJ$ must hold, that is, each object referenced by o must exist;*
- EXCLUSIVENESS*: for each object $o' \in comp(o)$, for each $r \in in\_paths(o')$ such that $composite(r) = true$*
  - *if $in\_rel(r) = false$, then for each $r' \in in\_paths(o')$ such that $composite(r') = true$, $r'^{-1}(o') \notin [\![exclusive(r)]\!]^+$ must hold;*
  - *if $in\_rel(r) = true$, then $\{o^* \mid r(o^*) = o' \wedge o^* \in [\![exclusive(r)]\!]^+\} = \{o\}$ must hold.*

□

The modification of an object can be dealt with as the creation of a new object. We recall that the OID of an object cannot be modified and that the modification of an object cannot result in object deletion. Therefore, the only constraints potentially violated by a modification are those related to exclusiveness and referential integrity. If those constraints are violated, the modification is simply disallowed (backed out).

Deletion

In this section we deal with object deletion. As we will see, object deletion must be carefully handled in presence of composite objects, because semantic ambiguities due to the different options may arise. In formulating the strategies for solving those ambiguities we follow the basic idea of preserving the information contained in the database from non explicitly requested deletion, and thus, in ambiguous cases, we have chosen not to delete an object.

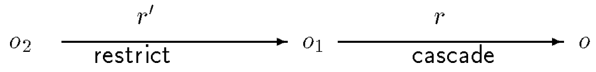

Figure 2: Objects and relationships of Example 3

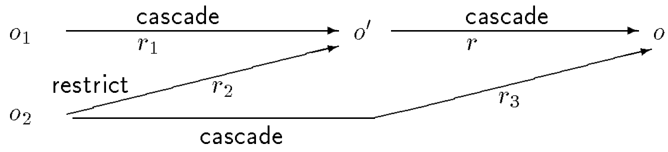

Figure 3: Objects and relationships of Example 4

Let us first show, by means of examples, why deletion may result in ambiguities, and then present the semantics we have defined to capture the desired behavior.

Example 3 *Consider the objects and the relationships in Figure 2. When object o is deleted, $delete(r)$ =* cascade *requires the deletion of o_1. By contrast, $delete(r')$ =* restrict *states that o_1 cannot be deleted. Thus, an ambiguity arises. We adopt a conservative approach and thus we choose not to delete object o_1. Thus, the* restrict *option of r' prevails on the* cascade *option of r.* ◇

Example 4 *Consider the objects and the relationships in Figure 3. When object o is deleted, two different possible final states can be obtained depending on the order in which traversal paths leading to o are considered.*

- *If r_3 is considered first, o_2 is deleted, since $delete(r_3)$ =* cascade *(note that, as a consequence, the traversal path from o_2 to o' is also removed). Then, r is considered and o' and o_1 are also deleted, because $delete(r) = delete(r_1)$ =* cascade*. Thus, all the four objects are deleted.*
- *If r is considered first, o' is not deleted because it participates in a relationship (r_2) with $delete(r_2)$ =* restrict*. Then, r_3 is considered and o_2 is deleted because $delete(r_3)$ =* cascade*. This deletion does not cause the deletion of any other object. Thus, the deletion of o only causes the deletion of o_2.*

Note that the final state which more closely reflects the intuitive semantics is the first one, because the object preventing the deletion of o', namely o_2, is in turn deleted as a consequence of the deletion of o. ◇

A similar situation may arise because of dependent declarations, as shown by the following example.

Example 5 *Consider the objects and the relationships in Figure 4. When object o is deleted, two different possible final states can be obtained depending on the order in which traversal paths originating from o are considered.*

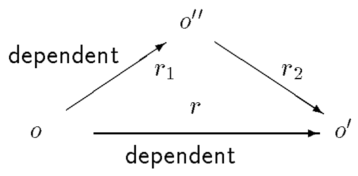

Figure 4: Objects and relationships of Example 5

- *If r_1 is considered first, o'' is deleted, because $dependent(r_1)$ =* true *and no other traversal path leading to o'' exists (note that, as a consequence, the traversal path from o'' to o' is also removed). Then, r is considered and o' is also deleted, because $dependent(r)$ =* true *and no more traversal paths leading to o' exist. Thus, all three objects are deleted.*
- *If r is considered first, o' is not deleted because it participates in relationship r_2. Then, r_1 is considered and o'' is deleted. Thus, the deletion of o only causes the deletion of o''.*

Note that the final state which more closely reflects the intuitive semantics is the first one, because the object to which o' belongs as a component (i.e. o'') is in turn deleted as a consequence of the deletion of o. $\diamond$

It is thus important to establish a well-founded semantics for delete operations on composite objects. Such semantics is defined in what follows. We start from the idea that, given an object o, we want to determine the set, denoted as Set^d, of the objects that have to be deleted as a consequence of the deletion of o. Obviously, before determining Set^d, it must be verified whether object o itself can be deleted, that is, whether no traversal path r leading to o[3] exists, such that $delete(r)$ = restrict.

The set Set^d initially contains only the object to be deleted, that is, $Set^d = \{o\}$. Other objects are then inserted into this set, because of dependent declarations or delete declarations, according to what follows.

- dependent declarations:

  Each object o' participating in some composite relationships with an object o in Set^d such that all the traversal paths leading to o' originate from objects in Set^d or from object o' itself, and at least one of these paths has a dependent declaration, is added to Set^d. Formally,

  $\forall o \in Set^d$ if $\exists\ r$ such that $r(o) = o'$, $dependent(r) = \mathsf{true}$, $in\_paths(o') = \{r_1, \ldots, r_n\}$ and $\forall i, i = 1, \ldots, n$, $r_i^{-1}(o') \in Set^d$ or $r_i^{-1}(o') = o'$, then $Set^d = Set^d \cup \{o'\}$.

- delete declarations:

  Each object o' participating in some composite relationships with an object o in Set^d through a traversal path r with $delete(r)$ = cascade and such that for all other traversal paths r' leading to o' either $delete(r') \neq$ restrict or r' originates from some object in Set^d or from object o' itself, is added to Set^d. Formally,

  $\forall o \in Set^d$ if $\exists\ r$ such that $r(o') = o$, $delete(r) = \mathsf{cascade}$, $in\_paths(o') = \{r_1, \ldots, r_n\}$ and $\forall i, i = 1, \ldots, n$, either $delete(r_i) \neq \mathsf{restrict}$, or $r_i^{-1}(o') \in Set^d$ or $r_i^{-1}(o') = o'$, then $Set^d = Set^d \cup \{o'\}$.

Now we introduce two operators formalizing the above concepts.

Definition 3 ($\vdash^d$). *Let o, $o' \in OBJ$ be two objects. o' is said to be deleted for dependence from o, denoted as $o \vdash^d o'$, if $r(o) = o'$, $dependent(r)$ =* true *and for any other relationship r' such that $o' = r'(o^*)$ either $o^* = o'$ or $o^* \in Set^d$.* □

Definition 4 ($\vdash^c$). *Let o, $o' \in OBJ$ be two objects. o' is said to be deleted for referential integrity from o, denoted as $o \vdash^c o'$, if $r(o') = o$, $delete(r)$ =* cascade *and for any other relationship r' such that $o' = r'(o^*)$ and $delete(r')$ =* restrict *either $o^* = o'$ or $o^* \in Set^d$.* □

We now introduce an operator which is the union of $\vdash^d$ and $\vdash^c$.

Definition 5 (DEL). *Operator* DEL : $\mathcal{P}(OBJ) \rightarrow \mathcal{P}(OBJ)$ *denotes the operator which returns the objects to be deleted (either for dependence or for referential integrity) from a given set of objects. Given a set of objects $O \subseteq \mathcal{P}(OBJ)$*

$$\begin{aligned}\mathsf{DEL}(O) = O\ & \cup \{o' \mid \exists\, o \in O \quad o \vdash^d o'\} \\ & \cup \{o' \mid \exists\, o \in O \quad o \vdash^c o'\}.\end{aligned}$$

□

We are now able to formally specify the semantics of the delete operation, relying on the notion of *fixpoint*. According to [6], given a poset $(A, \leq)$, where A is a set and $\leq$ is a partial order on A, and given a monotonic transformation T on the complete poset $(A, \leq)$, T has a least fixpoint. In our context, $(A, \leq)$ is $(\mathcal{P}(OBJ)^o, \subseteq)$, where $\mathcal{P}(OBJ)^o$ is the set of all possible sets of objects containing object o[4], "$\subseteq$" is set inclusion on $\mathcal{P}(OBJ)^o$ and the monotonic transformation T is DEL. $(\mathcal{P}(OBJ)^o, \subseteq)$ is a complete poset since for each subset O of $\mathcal{P}(OBJ)^o$ both the least upper bound (lub(O)) and

[3] This means that an object o' exists such that $r(o') = o$.

[4] $\mathcal{P}(OBJ)^o$ consists of the sets of objects O belonging to $\mathcal{P}(OBJ)$ such that $o \in O$.

the greatest lower bound (glb(O)) exist. Finally, DEL is a monotonic operator on $(\mathcal{P}(OBJ)^o, \subseteq)$. Indeed, for each $I, J \in \mathcal{P}(OBJ)^o$, $I \subseteq J$, $\mathsf{DEL}(I) \subseteq \mathsf{DEL}(J)$, because the DEL operator applied to set J (where J contains all the objects in I and eventually some other objects) returns all the objects in $\mathsf{DEL}(I)$ and eventually some more objects that can be deleted starting from J. Thus, the monotonic operator DEL has a least fixpoint, denoted as $lfp(\mathsf{DEL}) = Set^d$. An element $O \in \mathcal{P}(OBJ)^o$ is said a *fixpoint* of DEL if $\mathsf{DEL}(O) = O$.

Definition 6 *Let $o \in OBJ$ be an object, the effect of the deletion of o in a composite object model is the set $Set^d \in \mathcal{P}(OBJ)^o$ such that $Set^d = lfp(\mathsf{DEL})$.* □

The following example shows how the defined semantics correctly handles the "ambiguous" situation of Example 4.

Example 6 *Referring to objects and relationships of Example 4, the various iterations of the fixpoint computation lead to the following sets:*

$Set^d = \{o\}$	
$Set^d = \{o, o_2\}$	*for* $o \vdash^c o_2$
$Set^d = \{o, o_2, o'\}$	*for* $o \vdash^c o'$
$Set^d = \{o, o_2, o', o_1\}$	*for* $o' \vdash^c o_1$

The set $\{o, o_2, o', o_1\}$ is the effect of the deletion of o, because $\mathsf{DEL}(\{o, o_2, o', o_1\}) = \{o, o_2, o', o_1\}$. In the fixpoint computation, note that at the second iteration, o' has not been inserted in Set^d because of an incoming traversal path to it with delete restrict *(i.e. r_2), whereas in the third iteration o' is inserted because the origin of that traversal path (i.e. o_2) belongs to Set^d.* ◇

The following example illustrates the semantics of deletion in a more complex example.

Example 7 *Consider the objects and the relationships in Figure 5. Suppose that object o is deleted. The various iterations of the fixpoint computation lead to the following sets:*

$Set^d = \{o\}$	
$Set^d = \{o, o_1\}$	*for* $o \vdash^c o_1$
$Set^d = \{o, o_1, o_2, o_3\}$	*for* $o_1 \vdash^c o_3$ *and* $o_1 \vdash^d o_2$
$Set^d = \{o, o_1, o_2, o_3, o_6\}$	*for* $o_3 \vdash^c o_6$

The set $\{o, o_1, o_2, o_3, o_6\}$ is the effect of the deletion of o, because $\mathsf{DEL}(\{o, o_1, o_2, o_3, o_6\}) = \{o, o_1, o_2, o_3, o_6\}$.

This correctly models the semantics specified for the relationships. Indeed:

- *object o can be deleted, because there is no traversal path with delete* restrict *leading to it;*
- *as a consequence of the deletion of object o, object o_1 is deleted, because of a traversal path with delete* cascade *leading to o, and because no traversal path with delete* restrict *leads to o_1;*

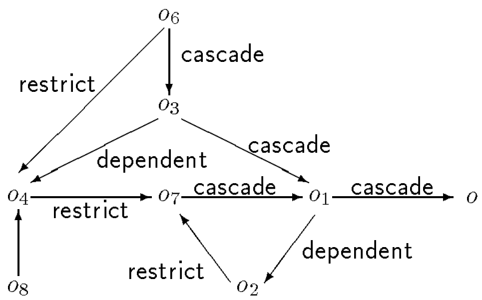

Figure 5: Objects and relationships of Example 7

- *as a consequence of the deletion of object o_1, objects o_3 and o_7 (due to a traversal path with delete* cascade *leading to o_1) and object o_2 (due to a composite dependent traversal path coming from o_1) could be deleted;*
  - *o_2 is deleted, because there are no more traversal paths leading to it;*
  - *o_3 is deleted, because there are no incoming traversal paths with delete* restrict*;*
  - *o_7 is not deleted, because there is an incoming traversal path (from o_4) with delete* restrict*;*
- *the deletion of o_2 does not cause the deletion of any other object, whereas the deletion of o_3 could cause the deletion of o_6 (due to a traversal path with delete* cascade *leading to o_3) and of o_4 (due to a composite dependent traversal path coming from o_3);*
  - *o_6 is deleted, because there are no incoming traversal paths with delete* restrict*;*
  - *o_4 is not deleted, because there is a traversal path leading to it (from object o_8);*
- *the deletion of o_6 does not cause the deletion of any other object.*

◇

CONCLUSIONS

In this paper we have proposed a composite object model. The proposed model supports both exclusiveness (with various degrees of granularity) and dependency specifications, as well as the declaration of the behavior to be taken upon violations of referential integrity. Though, for the sake of simplicity, in the paper we have only dealt with single-valued (that is, one-to-one) relationships, the model also covers multi-valued (that is, one-to-many and many-to-many) relationships. Furthermore, the model can be easily extended to associate composite declarations and referential integrity specifications with attributes.

We remark that our model expresses an important class of constraints related to the aggregation relationship among objects, in a declarative way. If such a specific handling of such a category of constraints were not provided, these constraints would have to be enforced by inserting repairing code in methods. An ad-hoc support for such important class of constraints seems, however, more adequate.

In the paper we have presented a language for specifying composite objects and have revised the semantics of update operations to take into account those constraints. However, though composite objects are useful from a semantic point of view, composite object handling must not decrease too much the performance of the system.

Our composite object model has been implemented in the Ode active OODBMS [1, 2]. Our approach relies on the use of a classical facility of DBMS to maintain data integrity, namely triggers [10]. In particular, we have developed a tool, automatically generating, from composite object specifications, a set of triggers enforcing the associated constraints. The use of triggers for supporting referential integrity has been investigated by Baralis et al. [4] whoc have shown how triggers can be derived for maintaining SQL referential integrity. However, they have not addressed how to solve the non-determinism that may arise from conflicting *restrict* and *cascade* options. In our tool, by contrast, problems related to non-determinism are solved according to the well-defined semantics we have described in the paper. This semantics could also be adopted in the relational context. Our tool, moreover, also appropriately handles exclusiveness and dependency constraints.

Some preliminary evaluations have been performed on the prototype implementation, to verify the practicability of our approach. While the results in terms of the general utility of the tool from a semantic modelling point of view and of ease of use are encouraging, it is more difficult to evaluate the prototype from the performance point of view, mainly because the Ode implementation itself, and in particular its trigger support, are not optimized.

We thus plan to implement the proposed model on some commercial OODBMS without active capabilities. In particular, we would like to investigate the possibility of exploiting appropriate access structure to efficiently support operations on composite objects. This means that operations on composite objects can be implemented by appropriately traversing the object aggregation graph. In such traversals, ad-hoc access techniques can be exploited, possibly adapted from those proposed in the relational context by Härder and Reinert [16]. Such an approach still needs further investigation.

REFERENCES

[1] R. Agrawal and N. Gehani. ODE (Object Database and Environment): The Language and the Data Model. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 36–45, 1989.

[2] R. Arlein, J. Gava, N. Gehani, and D. Lieuwen. *Ode 4.1 User Manual.* AT&T Bell Laboratories, Murray Hill, New Jersey, 1995.

[3] A. Artale, E. Franconi, N. Guarino, and L. Pazzi. Part-whole Relations in Object-Centered Systems: An Overview. *Data and Knowledge Engineering*, 20(3):347–383, 1996.

[4] E. Baralis, S. Ceri, and S. Paraboschi. Declarative Specification of Constraint Maintenance. In P. Loucopoulos, editor, *Proc. Thirteenth Int'l Conf. on the Entity-Relationship Approach*, number 881 in Lecture Notes in Computer Science, pages 205–222, 1994.

[5] E. Bertino and S. Jajodia. Modeling Multilevel Entities using Single-Level Objects. In S. Tsur, S. Ceri, and K. Tanaka, editors, *Proc. Third Int'l Conf. on Deductive and Object-Oriented Databases*, number 760 in Lecture Notes in Computer Science, pages 415–428, 1993.

[6] G. Birkhoff. Lattice Theory. *American Mathematical Society Colloquium Publications*, 25, 1973.

[7] R. Breitl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, and M. Williams. The GemStone Data Management System. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databasases, and Applications*, pages 283–308. Addison-Wesley, 1989.

[8] S.J. Cannan and G.A.M. Otten. *SQL - The Standard Handbook.* McGraw-Hill, 1992.

[9] R. Cattel. *The Object Database Standard: ODMG-93.* Morgan-Kaufmann, 1996.

[10] S. Ceri and J. Widom. *Active Database Systems - Triggers and Rules for Advanced Database Processing.* Morgan-Kaufmann, 1996.

[11] D. Chamberlin. *Using the New DB2 - IBM's Object-Relational Database System.* Morgan-Kaufmann, 1996.

[12] F. Civello. Roles for Composite Objects in Object-Oriented Analysis and Design. In *Proc. Eighth Int'l Conf. on Object-Oriented Programming: Systems, Languages, and Applications*, pages 376–385, 1993.

[13] O. Deux et al. The Story of o_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91–108, 1990.

[14] M. Halper, J. Geller, Y. Perl, and W. Klas. Integrating a Part Relationship into an Open OODB System using Metaclasses. In N.R. Adam, B.K. Bhargava, and Y. Yesha, editors, *Proc. of the Third Int'l Conf. on Information and Knowledge Management*, pages 10–17, 1994.

[15] E. Hanson. Rule Condition Testing and Action Execution in Ariel. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 49–58, 1992.

[16] T. Harder and J. Reinert. Access Path Support for Referential Integrity in SQL2. *VLDB Journal*, 5:196–214, 1996.

[17] Illustra Information Technologies, Oakland, California. *Illustra User's Guide*. Release 2.1.

[18] W. Kim, J. Banerjee, H.T. Chou, J.F. Garza, and D. Woelk. Composite object support in a Object-Oriented Database System. In N. Meyrowitz, editor, *Proc. Second Int'l Conf. on Object-Oriented Programming: Systems, Languages, and Applications*, pages 118–125, 1987.

[19] W. Kim, E. Bertino, and J. Garza. Composite Objects Revisited. In J. Clifford, B. Lindsay, and D. Maier, editors, *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 337–347, 1989.

[20] M. Kolp and A. Pirotte. An Aggregation Model and its C++ Implementation. In M.E. Orlowska and R. Zicari, editors, *Proc. of the Fourth Int'l Conf. on Object-Oriented Information Systems Engineering*, number 685 in Lecture Notes in Computer Science, pages 352–373, 1993.

[21] V.M. Markowitz. Safe Referential Structures in Relational Databases. In G. M. Lohman, A. Sernadas, and R. Camps, editors, *Proc. Seventeenth Int'l Conf. on Very Large Data Bases*, pages 123–132, 1991.

[22] J. Melton and A.R. Simon. *Understanding the New SQL: a Complete Guide*. Morgan-Kaufmann, 1993.

[23] R. Motschnig-Pitrik. The Semantics of Parts versus Aggregates in Data/Knowledge Modelling. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proc. of the Fifth Int'l Conf. on Advanced Information Systems*, pages 211–224, 1997.

[24] C. Peltason, A. Schmiedel, C. Kindermann, and J. Quantz. The BACK System Revisited. Technical Report KIT - Report 75, Technische Universitat Berlin, 1989.

[25] M. E. Winston, R. Chaffin, and D. Herrmann. A Taxonomy of Part-whole Relations. *Cognitive Science: a Multidisciplinary Journal of Artificial Intelligence, Linguistics, Neuroscience, Philosophy, Psychology*, 11(4):417–442, 1987.