An Extension of the ODMG Data Model to Support Ternary Relationships

ESPERANZA MARCOS
Sciences and technology Department
University Rey Juan Carlos
C/ Tulipan s/n, 28933-Móstoles (Madrid)
SPAIN

BELÉN VELA
Computer Sciences Department
Univ. Carlos III of Madrid
C/ Butarque s/n 28911- Leganés (Madrid)
SPAIN

Abstract: - It is well known that the object data model has improved the expression power with regard to the relational model, supporting constructors such as multi-valued attributes or inheritance. However, there are still some other important constructors, as ternary relationships or composite objects, that are neglected by most of the object data base systems, even by the object database standard, ODMG. Nevertheless, ternary relationships (and n-ary relationships, in general) are supported by almost every conceptual model due to their importance in modelling the real world. The problem is that n-ary relationships can only be used to define the conceptual schema, and so, it is necessary to invent a way of covering the gap between n-ary relationships allowed at a conceptual level and binary ones supported by the implementation models. To solve the mentioned problem, we propose an extension of the ODMG data model to support ternary relationships, based on the definition of a new collection type.

Keywords: - Relationships, ternary relationships, object-oriented database systems, data models, collection types, ODMG. IMACS/IEEE CSCC'99 Proceedings, Pages:7211-7218

1. Introduction

One of the most important improvements of the new generation databases (object-oriented and object-relational data bases) with regard to the relational ones is their capacity to support complex objects and structures. In order to achieve this goal, database models have improved their expressiveness through new primitives being now closer to the conceptual models than the relational model was. However, some important conceptual constructors, as for example, aggregation or n-ary relationships, are not still supported by the database models, being necessary to maintain its semantics through the applications.

Perhaps one of the most important constructors to represent the real world is the relationship, which allows representing the relationship concept that is present at almost every application. In spite of several proposals extending the relationship semantics [1, 4, 11] to support different meanings of the relationship concept, currently, database models, including the ODMG data model, just support binary relationships [5, 8, 13].

In this paper, we propose an ODMG extension to support ternary relationships through a new collection type that we have called *PairColl*. This proposal can be easily extended to support n-ary relationships.

The remainder of the paper is organised as follows. First, in section 2, we justify the need of the ternary relationships in data models. Afterwards, in section 3, we summarise the current state of the relationships and collection types in the ODMG data model. Then, in section 4, we explain our proposal. Finally, in section 5, we conclude pointing out some possible extensions and future works.

2. Modelling with ternary relationships

A relationship is an association between two or more objects that belong in general, but not mandatory, to different object types. When only two objects participate in the association it is called a binary relationship. However, it is very common in the real world to find associations that relate three or more objects.

In such cases we speak of n-ary relationships. A ternary relationship is an special case of n-ary relationship, in which *n* is three.

Ternary, and n-ary relationships in general, are very important constructors in a model to represent the semantics of the real world. Nearly all conceptual models, as E/R [7], UML [2], MIMO [12], Merise [14], etc. support ternary relationships. However, none of the current database standards, the ODMG [5], the SQL92 [13] as well as the future standard SQL3 [8], supports this constructor. Moreover, implementation models, neither object-relational (as DB2 [6] or Oracle [15]) nor pure object-oriented models (as GemStone [3], Orion [10] or POET [16]) support ternary relationships.

Thus, ternary relationships can be directly represented in a database conceptual schema but they cannot be represented in an implementation one. For this reason, it is necessary to define how ternary relationships should be implemented. There are two different ways to broach this problem:

a) The first one is to convert the relationship in an object type (or in a table if we are working with a relational product). This solution forces to represent ternary relationships, unlike the binary ones, as object types. However, this is not a good solution because in addition to the loss of information, which is due to considering a relationship as an object type, it also gives rise to a more complex schema (notice that the addition of a new object type involves adding three binary relationships).
b) The second possible solution is to break off the ternary relationship representing it with two or three binary relationships, in accordance with the cardinalities. However, this solution is not always applicable because some ternary relationships cannot be represented as binary relationships without some loss of information [9, 17]. If we can represent a ternary relationship through some binary ones then the relationship is conceptually not really a ternary one, although it had been considered so in a first approach of the conceptual schema. If the association is really a ternary relationship then it cannot be broken off without some loss of information.

Figure 1(a) shows an example of ternary relationship. In this example a teacher can *teach* the same subject in one or more course/s. In the same way, a subject in a course can be *taught by* one or more teacher/s. And, finally a teacher *teaches* in a course one or more subjects. This association could be represented with three binary relationships as it is shown in figure 1(b).

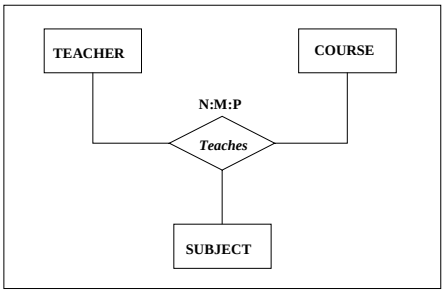

Figure 1(a): An example of a ternary relationship

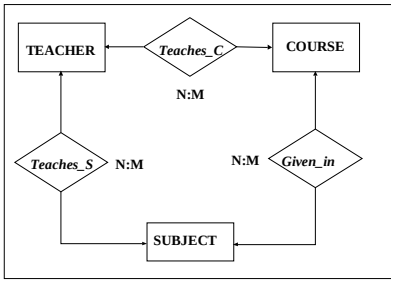

Figure 1(b): An example of a ternary relationship broken off

However, figures 1(a) and 1(b) represent two schemata that are not equivalent between them. If we represent the relationship *Teaches* as it is shown in figure 1(b) we would be able to know, for example, that Alan Smith teaches maths and history or that he teaches in the first and in the second course. However, we would not be able to know which matter Alan Smith teaches in each course. So, the complete information represented by the ternary relationship is lost when the ternary relationship is broken off into three binary relationships and we are not able to know which matter is taught by each teacher in each course. Due to the fact that such cases can be easily found in the real world, it is important that the standard and implementation models can directly support them.

3. Relationships in ODMG 2.0

The ODMG standard is the object database standard proposed by the ODMG (Object Data Management Group). ODMG defines an Object Model and Object Definition Language (ODL) that supports this model. It also provides an Object Query Language (OQL) and the C++, Smalltalk and Java ODL bindings.

The ODMG object model supports only binary relationships. A relationship is defined implicitly by the definition of traversal paths that are declared in pairs, one for each direction of the binary relationship. A traversal path definition includes the target type and information about the inverse traversal path found in the target type. 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. The multiple cardinality of the target type is supported through collection types. If no collection type is used, the cardinality on the target side is one. Figure 2 shows an example of a many-to-many binary relationship in ODL.

```
class teacher                          class course
  { attribute string (30) name;          { attribute string (30) description;
    attribute string (20) matter;          attribute short n_hours;
    relationship set<course> teaches       relationship set <teacher> taught_by
        inverse taught_by::course;             inverse teaches::teacher;
    void teacher ();                       void course();
    void ~teacher ();};                    void ~course ();};
```

Figure 2: An example of a binary relationship in ODL

The ODMG data model supports the following collection types: *set*, *list*, *bag*, *array* and *dictionary*. However, only the set, list and bag are allowed in the relationship definition. Figure 3 shows the relationship specification in the ODL grammar, as well as the main part of the specification for the collection types.

```
<rel_dcl> ::= relationship
    <target_of_path>
    <identifier>
    inverse <inverse_traversal_path>
<target_of_path>::= <identifier>
    | <rel_collection_type> <<identifier>>
<inverse_traversal_path>::=
          <identifier>::<identifier>
<rel_collection_type>::= set | list | bag

<coll_type>::=
    <coll_spec> <<simple_type_spec>>
    | dictionary <<simple_type_spec>,
                  <simple_type_spec>>
<coll_spec>::= set | list | bag
```

Figure 3.- ODL grammar for binary relationships and for collection types

Set, *list*, *bag*, *array* and *dictionary* type are defined in the ODL meta-schema as subtypes of the *collection* meta-type, which itself is defined as a subtype of the *object* meta-type. Despite both, the array and the dictionary type, are supported by the ODMG data model as subtypes of the collection type, we can see that they cannot be used to define relationships. Our proposal consists on defining a new collection type, similar to the dictionary one, and allowing to use it in the relationship definition. The idea of implementing relationships as collections of pairs is also used in other data models but with a different approach. So, for example, in the Object Data Model OM [18] relationships are represented by a special form of collection, called *binary* collection, in which each element is a pair identifying the related objects. The *binary* collections of OM are similar to the *PairColl* collections that we propose. Nevertheless, whereas a binary collection can represent only binary

relationships, a PairColl collection will be used to represent ternary relationships.

4. Extending the ODMG data model to support ternary relationships

There are different approaches to support ternary relationships in a data model:

1) The first one is to support ternary relationships by a *star configuration*, see figure 4(a), considering relationships as "first category" objects which relate different object types [4, 11].
2) The second approach is representing ternary relationships by a *ring configuration*, see figure 4(b), through links or traversal paths where each object type is related to every object type involved in the relationship. Notice that this solution is different to represent ternary relationships as three binary ones, because in a ring configuration each object is simultaneously related with two other objects. For example, think in a ring of three children taken by their hands; all of them are joined together in a ternary relationship rather than in three binary ones.

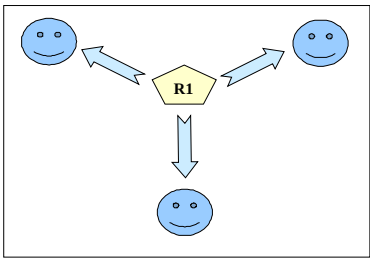

Figure 4(a): A ternary relationship supported by a star configuration

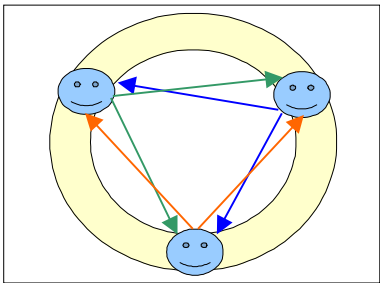

Figure 4(b): A ternary relationship supported by a ring configuration

The star configuration allows a better representation of the real world because of the semantics that it is able to support. For example, this configuration allows representing relationships, either binary or n-ary, with their own attributes or even, their own relationships [12, 14]. However, a ring configuration does not allow representing directly relationships with attributes and, when a relationship has attributes, it is necessary to define a new object type to represent it. Despite the efficiency depends on the implementation, in some queries the ring configuration could be more efficient because each object keeps directly the references to the objects with which it is related.

In spite of the advantages and disadvantages of each alternative, we have chosen the ring configuration in order to keep the ODMG philosophy because this is the way in which the ODMG data model represents the binary relationships. Defining ternary relationships by a star configuration will force us to redefine the binary relationships model in the ODMG data model introducing the relationship concept as a "first category" object.

Thus, ternary relationships can be defined as an association which, unlike the binary relationships, relate more than two object types. In order to support them we must introduce the following changes to the ODMG data model:

a) Defining a new collection type, called ***PairColl***, by a new interfaz. Moreover we must to extend the ODL grammar, in a similar way as the dictionary type, in order to allow keeping collections of pairs of object types.
b) Defining a ***ternary relationship*** interface in the ODMG meta-schema, and modifying the ODL grammar to allow defining ternary relationships by the new PairColl collection type.
c) Studying the possible implications over the ***OQL***.

The extended ODMG and ODL will be called ODMG+ and ODL+ respectively.

4.1 The new PairColl collection type

The *PairColl* collection type should be similar to the dictionary type with some changes. The dictionary type is defined as "*an unordered*

sequence of key-value pairs with no duplicate keys. Each key-value pair is constructed as an instance of the following structure: struct Association {any key; any value;}" [5]. The dictionary type is not able to support ternary relationships because the key must be unique, and due to this fact a ternary relationship with *n:m:p* multiplicity cannot be represented. Thus, the *PairColl* should be defined as a dictionary type without the key restriction.

Definition:

*A **PairColl** object is an unordered collection of key-key pairs with no duplicate pairs. Each key-key pair is constructed as an instance of the following structure: struct Pair {any key; any key;}, through the PairColl <<v1, v2>> template.*

PairColl =

{<k1, k2> / k1, k2 ∈ T ∧ k1≠ k2, ∀ <k1, k2>}, where T is any valid ODMG type

A *PairColl* interface must be defined. It should be a subtype of the collection type, like the dictionary type. Figure 5 shows the *Collection* and *PairColl* interfaces definition.

```
interface Collection: Object {
    exception           InvalidCollectionType {};
    exception           ElementNotFound{any element;};
    unsigned long       cardinality();
    boolean             is_empty();
    boolean             is_ordered();
    boolean             allows_duplicates();
    boolean             contains_element(in any element);
    void                insert (in any element);
    void                remove (in any element)
                                raises (ElementNotFound);
    Iterator            create_iterator (in boolean stable);
    BidirectionalIterator  create_bidirectional_iterator (in boolean stable)
                                raises (InvalidCollectioType);};

interface PairColl: Collection{
    void        bind (in any k1, in any k2);
    void        unbind (in any k1, in any k2);
    void        find (in any k1, in any k2);};
```

Figure 5: PairColl interface definition in ODMG+

In addition to the operations defined in the Collection interface, collection objects, including *PairColl* objects, inherit the operations of the Object interface as, for example, comparison between two collections, copy of a collection, etc.

The bind, unbind and find operations create a pair from the two keys received as arguments. Afterwards, they call the insert_element, remove_element and contains_element in order to insert, remove or select an element in the *PairColl* respectively.

PairColl redefines the insert_element, remove_element and contains_element inherited from the Collection interface. These operations are valid for the *PairColl* when the specified argument is a *Pair* parameter generated by the correspondent bind, unbind and find operations. The insert_element operation, after the addition of the new object, checks that the object passed as an argument does not belong already to the set. The remove_element operation removes the element from the *PairColl* that matches the (v1, v2) pair passed as an argument. In the same way the search of a pair is done using the two values of the pair. Therefore, when the (v1, v2) pair passed as an argument matches some pair in the *PairColl* a true value is returned.

Once defined the *PairColl* type, we have to add a new production rule to the ODL Grammar (see figure 6) to include this new collection type. The modifications with regard to the ODL standard are introduced in bold type; the terminal symbols are introduced in italic type.

```
<coll_type>::=
    <coll_spec> <<simple_type_spec>>
    | dictionary <<simple_type_spec>, <simple_type_spec>>
    | paircoll <<simple_type_spec>,<simple_type_spec>>

<coll_spec>::= set | list | bag
```

Figure 6.- ODL+: extension of the grammar for defining the new collection type PairColl

4.2 Adding ternary relationship to the ODMG data model

The first step is to define the meta-schema extensions. A *TernaryRelationship* interface, which inherits from the *Property* interface, must be defined in the ODMG meta-schema.

Figure 7 shows the *TernaryRelationship* interface definition that is similar to the definition of the *Relationsip* interface

.

```
enum TernaryCardinality {c1_1_1, c1_1_N, c1_N_M, c1_N_1, cN_1_1, cN_M_1,
cN_1_M, cN_M_P};

interface TernaryRelationship: Property{
    exception              ternaryintegrityError {};
    ternaryrelationship    TernaryRelationship traversal
          inverse <TernaryRelationship::traversal, TernaryRelationship::traversal>;
    TernaryCardinality     getTernaryCardinality; };
```

Figure 7: ODMG+: TernaryRelationship interface definition in the meta-schema

To represent the traversal direction of the ternary relationship, three TernaryRelationships meta-objects are required. Operations to form and drop the ternary relationship, as well access operations for manipulating its traversals, are defined implicitly, just as they are defined for the *Relationship* interface. It could be also considered the possibility of introducing in the ODMG meta-schema a new interface. In this case, *Relationship* and *TernaryRelationship* would be defined as subtypes of this new interface.

Despite the TernaryCardinality type defines all possible combinations, by the moment we have only considered ternary relationships whit n:m:p maximum cardinalities.

Once defined the *TernaryRelationship* interface, we must also introduce the definition of the ternary relationship in the ODL Grammar. Thus, the ODL specification shown in the figure 3 will be transformed in the specification of the figure 8, where a <rel_dcl> can be a binary or a ternary relationship respectively defined through the <binary_rel> or the <ternary_rel> specification.

```
<rel_dcl>::= <binary_rel>|<ternary_rel>
<binary_rel>::= relationship <target_of_path> <identifier> inverse <inverse_traversal_path>
<target_of_path>::= <identifier>| <rel_collection_type> <<identifier>>
<inverse_traversal_path>::= <identifier>::<identifier>
<rel_collection_type>::= set | list | bag
<ternary_rel>::= ternaryrelationship <ternary_target_of_path> <identifier>
                        inverse <ternary_inverse_traversal_path>
<ternary_target_of_path>::=  <ternary_rel_collection_type> <<identifier>, <identifier>>
<ternary_inverse_traversal_path>::= < <identifier>:: <identifier>, <identifier>:: <identifier> >
<ternary_rel_collection_type >::= paircoll
```

Figure 8.- ODL+: extension of the ODL grammar with ternary relationships

The definition of the binary relationships has not been modified. The ternary relationships specification defines the <ternary_ target_ of_ path> and an <identifier>, which is the name of the ternary relationship, as well as the <ternary_ inverse_ traversal_ path>. The <ternary_ target_ of_ path> specification defines a <ternary_ rel_ collection_ type>, which is a *PairColl* specification. The <ternary_ inverse_ traversal_ path> defines two pairs <identifier>:: <identifier>, one for each object type involved in the relationships. The <identifier> placed before the colon refers to the related object type, whereas the <identifier> placed after the colon is the name of the relationship in the related object type.
Figure 9 shows, as an example, the ODL definition of the ternary relationship taken from figure 1.

```
class Teacher (extend teachers)
{...ternaryrelationship paircoll <Course, Matter> teaches
        inverse < <Course>:: taught_by, <Matter>::given_by >;
 ...};
class Course (extend courses)
{... ternaryrelationship paircoll <Teacher, Matter> taught_by
        inverse < <Teacher>:: teachs, <Matter>::gived_by >;
...};
class Matter (extend matters)
{... ternaryrelationship paircoll <Teacher, Course> gived_by
        inverse < <Teacher>:: teach, <Course>::taugh_by >;
 ...};
```

Figure 9.- A ternary relationships example in ODL+

4.3 Implications over the OQL

To navigate through traversal paths, the OQL provides the *dot* operator which has two different notations, "." and "->", equivalent between them. The OQL syntax to navigate trought related objects is the following:

query ::= query dot relationship_name
dot ::= . | ->

According to the new ODL proposal, the *dot* operator should permit navigating through double traversal paths in order to retrieve the two objects related with a specific object by a ternary relationship.
Thus, the following query,

select t.teaches
from t in teachers
where t.name= "Alan Smith"

retrieves a literal *PairColl <Course, Matter>* containing the *courses* and *matters* taught in each course by Alan Smith. So, now we are able to know that Alan Smith teaches maths in the first course and history in the second course.
Navegating through a double transversal path complicates the accesing to the attributes of the related objects. This is possible but rather tedious, requiring a nested 'select-from-where'. It could be also convinient to introduce some extension to the OQL in order to simplify this kind of querys.

5. Conclusions and Future Works

In this paper we have proposed an extension of the ODMG data model to support ternary relationships, through a new collection type called *PairColl*. In addition, we have redefined the ODL grammar in order to include the *PairColl* type definition and the ternary relationship definition. Some of the OQL implications have also been considered.
Our proposal has still some weaknesses that we have to consider in our next works. Therefore, we should extend the OQL to allow retrieving the attributes of the related objects without a nested 'select-from-where'. We have also to define some other OQL extensions as, for example, the PairColl constructor. We have also to extend the ODL proposal to support all possible maximum cardinalities (c1_1_N, c1_N_N, etc.) rather than only support n:m:p ternary relationships. Moreover, we will study

the convinience of extending this proposal to support n-ary relationships.

Now, we are working in a formal description of the ternary relationships and we are also implementing in C++ the *PairColl* type, using generic data types, and the ternary relationships (considering the referential integrity, consistency, etc.). Afterwards, we will extend the POET data model, which is a pure object-oriented database system based on C++ [16], including the *PairColl* type and ternary relationships.

References:

[1] Bertino and Guerrini, "Extending the ODMG Object Model with Composite Objects", *Proceedings of the 1998 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA'98)*, in ACM SIGPLAN Notices, Vol. 33, No. 10, Oct. 1998, pp.259-270.

[2] G. Booch, J. Rumbaugh and I. Jacobson, "Unified Modelling Language, V1.0", *Rational Software Corporation*, January 1997.

[3] Breitl, R., et al., "The GemStone data management system", in *Object-Oriented Concepts, Databases, and Applications*, pp. 283-308, W. Kim, and F. Lochovsky (eds.), Addison-Wesley, 1989.

[4] D. Calvanese and M. Lenzerini, "Making Object-Oriented Schemas More Expressive". *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (SIGMOD/PODS 94)*, Minneapolis, Minnesota USA, May 24-26, 1994, pp. 243-254.

[5] R.G.G. Cattell and Douglas K. Barry, "*The Object Database Standard: ODMG 2.0"*, Morgan Kaufmann Publishers, San Francisco 1997.

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

[7] P. Chen, "The Entity/Relationship Model: Toward a Unified View of Data". *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976.

[8] C.J.Date with Hugh Darwen, "A guide to the SQL standard". *Mc. Graw-Hill*, 1997.

[9] J. Dullea and Il-Y. Song, "An Analysis of the Structural Validity of Ternary Relationships in Entity Relationship Modeling", in Proc. of the Conference on Information Knowledge Management (CIKM'98), 1998.

[10] Kim, W., et al., "Features of the ORION object-oriented database system", in *Object-Oriented Concepts, Databases, and Applications*, pp. 251-282, W. Kim, and F. Lochovsky (eds.), Addison-Wesley, 1989.

[11] M. Kolp and A. Pirotte, "An Aggregation Model and its C++ Implementation". In *Proceedings of the Fourth International Conference on Object Oriented Information Systems*, Brisbone, Australia, 1994.

[12] E. Marcos, "MIMO: Propuesta de un Metamodelo de Objetos y su aplicación al diseño de bases de datos". *PhD. Thesis*, Universidad Politecnica de Madrid, 1997.

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

[14] Morejon, "MERISE: Vers une modélisation orientée object". *Les Éditions d'Organisation*, 1994.

[15] Oracle, "Objects and SQL in Oracle8". Oracle Technical White paper. Presented in the *Extended DataBase Technology conference (EDBT'98)*. Valencia (Madrid), Mars 1998.

[16] "POET/Programmer's Guide 4.0", *POET Software Corporation*, 1996.

[17] Il-Y. Song and T.H.Jones, "Ternary Relationship Decomposition Strategies Based on Binary Imposition Rule", IEEE 1995

[18] A. Steiner and M.C.Norrie, "Temporal Object Modelling". Proceedings of the *9th International Conference on Advanced Information Systems Engineering (CASE'97)*. Ed. A.Olivé and J. A. Pastor, Spring Verlag, 1997.