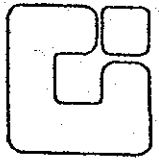

ER Institute

11th
International Conference
on the

Entity Relationship Approach

October 7-9, 1992
Karlsruhe, Germany

Advance Program

"Part" Relations for Object-Oriented Databases

Michael Halper, James Geller, and Yehoshua Perl

Institute for Integrated Systems, CIS Department
and
Center for Manufacturing Systems
New Jersey Institute of Technology
Newark, NJ 07102 USA

Abstract. It has long been recognized that the "part" relation is an extremely useful modeling tool. This is especially true in areas such as manufacturing, design, graphics, and document processing. In this paper, we present a comprehensive conceptual model for parts in the context of object-oriented database (OODB) systems. Our model captures the semantics and functionality of a variety of part relations with characteristics such as exclusiveness/sharing, multi-valuedness, cardinality restriction, ordering, essentiality, dependency, and value propagation. Our notion of exclusiveness extends previous work by refinement into two kinds, inter-class and intra-class exclusiveness. Dependency in our model is permitted from both the part to the whole object, and vice versa. We also present a general mechanism for upward and downward value propagation along the part relation. Of note is the fact that we realize the part model without having to introduce any extraordinary new constructs into the underlying OODB data model. The part relation itself is represented as an object class whose instances represent the actual part connections between instances of the participating classes. By elevating the part relation to the status of a "first-class citizen," we are following in the tradition of the ER and other semantic data models.

1 Introduction

The specialization (*is-a*) relation has long been the cornerstone of semantic [26] and object-oriented data models [2, 5, 10]. Another relation which has begun to receive considerable attention is the "part" relation. The need for part decomposition can be found in many advanced modeling domains, many of which have been targeted as testing grounds for object-oriented database (OODB) systems. In fact, a number of such systems have included intrinsic support for the part relation (e.g., [21, 23, 25, 29]). An example application is a database used in a manufacturing enterprise, where, after all, the main activity is the assembly of collections of parts into whole products. The part relation is also used extensively in CAD systems [22] and computer graphics [11].

Part decomposition has generated interest in Artificial Intelligence and related fields, where much attention has been paid to the part relation as it occurs in the context of logical syllogisms [31]. In particular, the question of the part relation's transitivity has been investigated [7]. The part relation has also been investigated in the context of connectionist networks (e.g., [18]).

In previous work [12, 16], we have investigated different kinds of part relations for graphical deep knowledge [13]. In this paper, we present a conceptual model for "parts" in the context of an OODB. This conceptual model (henceforth referred to as the *part model*) comprises a variety of part relations and their realization above an existing OODB data model. Characteristics of the different part relations include exclusiveness/sharing, multiplicity, cardinality restriction, ordering, essentiality, dependency, and upward/downward value propagation.

The ORION part model [21] distinguishes between four types of part relations, derived by imposing two types of constraints, exclusiveness and dependency, on *weak* references (what we call relationships). The *exclusiveness* constraint permits a "whole" object to put an exclusive hold on a part. In this paper, we further refine the exclusiveness property into two kinds, intra-class and inter-class exclusiveness. *Dependency* gives the schema designer the ability to make a part dependent on the existence of the whole: If the whole is deleted, the part is deleted automatically. Our model extends this idea and permits the specification of dependency in both directions, from the whole to the part, and vice versa. ORION allows a part relation to be set-valued. Our model includes the ability to impose cardinality constraints on the part relation. For example, we can say that a car has exactly four tires or a truck has between four and eighteen.

The part model of SNOOD [25] addresses the problem of value propagation (or selective inheritance) where values of certain attributes are made available at different levels of the part hierarchy. In our representation, we introduce a general mechanism for performing both upward and downward value propagation along the part relation.

An important aspect of our part model is the fact that we have avoided introducing any extraordinary new constructs into the underlying OODB model. We exploit features of OODBs which have been widely investigated and studied. Our model elevates part relations to the level of classes and objects: The part relation between two classes is represented as an object class in its own right, with its instances representing the actual part connections between pairs of objects from the participating classes [1, 8, 23, 28]. In this sense, our model is close to the ER [6] and other semantic models [26], where both entities and relationships are viewed as "first-class citizens."

The rest of the paper is organized as follows. In the next section, we discuss the underlying OODB model. In Section 3, we define the different part relations included in our model. Section 4 contains the details of incorporating these relations into the OODB. Conclusions and a discussion of future work are found in Section 5.

2 Preliminaries

In this section, we present features whose presence we assume in the underlying OODB model. While these features are common to existing OODBs, we closely follow the terminology of the Vodak Modeling Language [9] and the Dual Model [14, 15, 24]. An OODB relationship is a (named) property of an object which references another object. (At the schema level, we say that the relationship is a property of one class referring to another.) A relationship which does not refer to some object is said to be nil-valued.

A multi-valued relationship references a set of objects. Explicit upper and lower bounds can be placed on this set's cardinality [3], yielding a *range-restricted* relationship. We assume that our underlying OODB supports two other types of such constraining relationships, *essential* and *dependent*. An *essential relationship* is one which must always refer to an existent object (i.e., which may not be nil). It is equivalent to a range-restricted relationship whose upper and lower bounds are both equal to one. A *dependent relationship* from a class `A` to a class `B` exists if the existence of an instance of `A` depends on the existence of an instance of `B`. The deletion semantics for such a relationship is as follows: If an instance a of `A` references an instance b of `B`, and b is deleted, then a is also deleted.

A *path method* [14] is one which traverses a path through an object's composition hierarchy [20] to retrieve some data value. In other words, it is a sequence of messages passed along such a path. The concept is similar to the notion of *path expression* introduced in [5].

In [17], we introduced a graphical notation for the specification of OODB schemata, which will be employed throughout the paper. The conventions are: A class is represented as a box enclosing the name of the class (Fig. 5). An attribute is an ellipse which circumscribes a name. A single-valued relationship is denoted by a labeled, single-lined arrow pointing from the source class to the target class. A multi-valued relationship is similar, but the arrow is dual-lined (Fig. 11). A relationship is designated dependent (essential) with the addition of an extra arrow head (a circle). A path method is represented as a labeled, broken-lined arrow directed from the class of definition to the data item it accesses.

3 Definitions of the Part Relations

Our part model comprises a number of different part relations, defined with the following characteristics: exclusiveness/sharing, single-valuedness/multi-valuedness, cardinality range-restriction, ordering, and essentiality. There are also two part relations which express dependency semantics, and two others which permit value propagation along the part relation link.

Before getting to the specific definitions, let us define the terminology we will be using when discussing the part relations. Following Winston *et al.* [31], we will occasionally refer to the "part" relation as the *meronymic* relation. A part will be called a *meronym*, while the whole will be called the integral object or *holonym*. We will sometimes refer to the class of a part as the meronymic class; likewise, the class of the integral object will be referred to as the holonymic class. For example, an instance e of class **engine** would be a meronym and an instance c of class **car** would be a holonym. The classes, **engine** and **car**, would be the meronymic and holonymic classes, respectively.

Part relations in general can be divided along the lines of *exclusive* and *shared* [21, 25]. Designating a part relation exclusive means that the integral object is the sole owner of the part object. The need for exclusiveness arises when modeling *physical* (or *extensive*) objects [31] such as cars or buildings. In order to capture the semantics of such applications, the part relation must permit the explicit statement and enforcement of the fact, e.g., that cars do not share engines.

Part relations which are not exclusive are called *shared*. A shared part relation puts no restrictions on the way other objects can reference an object which is already part of some integral object. A part can be freely shared among several holonyms. For example, many memoranda may share the same textual body, or two books (compilations) may share the same chapter.

The exclusive/shared dichotomy is supported in a number of existing OODB part models (e.g., [21, 25]). In our model, we further refine the notion of exclusiveness by defining two kinds, *inter-class* and *intra-class* exclusiveness.

While it is the case that no two cars can share an engine, it is also the case that a car and an airplane cannot share one either. Therefore, the exclusive part relation between the classes `car` and `engine` has ramifications for the entire database topology, restricting not only "part" references from cars but from objects belonging to other classes as well. We call such a part relation *inter-class exclusive* because the reference restriction applies across all classes. There are times, however, when we would like to enforce the exclusive reference restriction only within a single class, and relax it otherwise. In other words, we want to be able to enforce the following: The fact that an object a of class `A` has a part reference to an object b of class `B` disallows any other instance of `A` from claiming b as its part, but does not disallow an instance of a class other than `A` from doing so. Let's look at an example where this restriction is relevant.

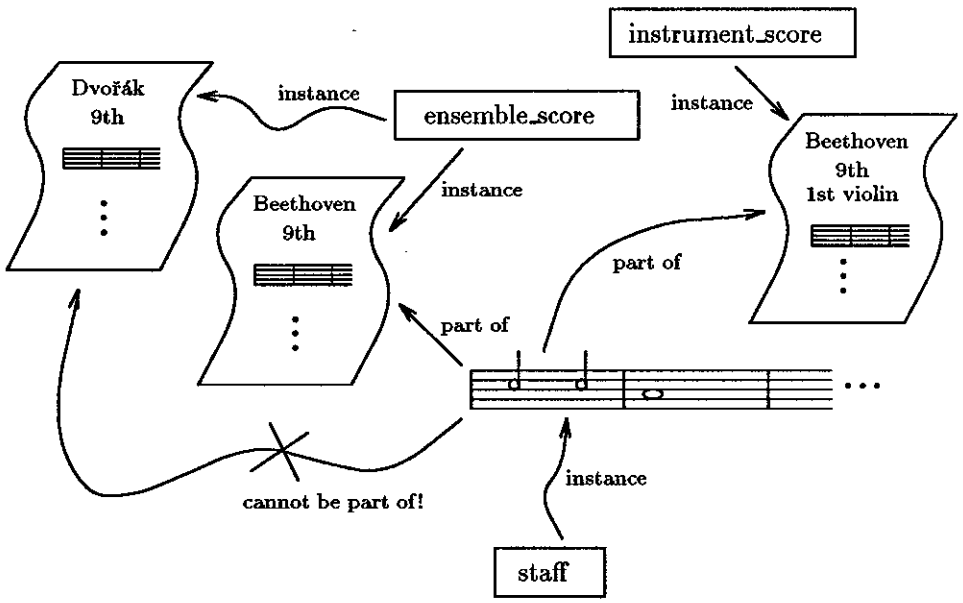

Fig. 1. Intra-class exclusive part relations in a music publication database

Consider a music publication system. The score for an orchestral composition is typically available in two formats, the full (or ensemble) score and individual instrument scores. The "staff object" representing the music to be played, e.g., by the first

violin section in Beethoven's Ninth Symphony can be modeled as part of both an ensemble score object and an instrument score object. However, it cannot be part of more than one ensemble score because different musical compositions do not have identical music played by the same instrument. For example, the music for the first violin section is not the same for Dvořák's Ninth as it is for Beethoven's Ninth (Fig. 1). Thus, the part relation between the classes `ensemble_score` and `staff` is intra-class exclusive. The same is true of the part relation between `instrument_score` and `staff`. See Fig. 1 for an illustration of this scenario.

Another example of the intra-class exclusive link is found between `ensemble_score` and `score_expression_sequence`. A score expression sequence is the line above the staff of a musical score containing annotations such as "Allegro" (Fig. 2). As tempo markings and other such performance notation vary from score to score, an expression sequence object will always be part of only a single ensemble score. In contrast, the same score expression sequence will constitute a part of all instrument scores associated with a particular ensemble score. Thus, the part relation between `ensemble_score` and `score_expression_sequence` is intra-class exclusive, while that between `instrument_score` and `score_expression_sequence` is shared.

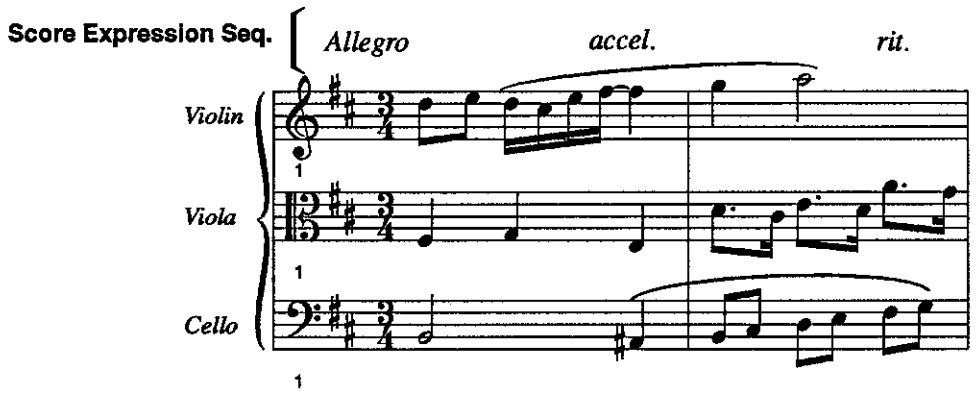

Fig. 2. An ensemble score and its score expression sequence

Single-valued part relations are those where the holonym can have only one meronym of the given type. For example, a car has only one engine. When holonyms can have many components of the same type, the relation is *multi-valued*, as with a car and its doors. To enhance expressiveness, our part model includes a number of variations of the multi-valued part relation.

The *range-restricted* part relation puts constraints on the number of meronyms that an integral object can have, allowing explicit upper and lower bounds to be placed on the relation's cardinality. For example, an engine can be required to have between four and twelve cylinders. The upper or lower bound may be omitted, indicating an "n or greater" or "0 to n" semantics.

The *fixed-cardinality* part relation is a special case of the range-restricted relation with upper and lower bounds of equal value. If only engines with six cylinders are

being modeled in the database, then a fixed-cardinality relation of degree six would be used.

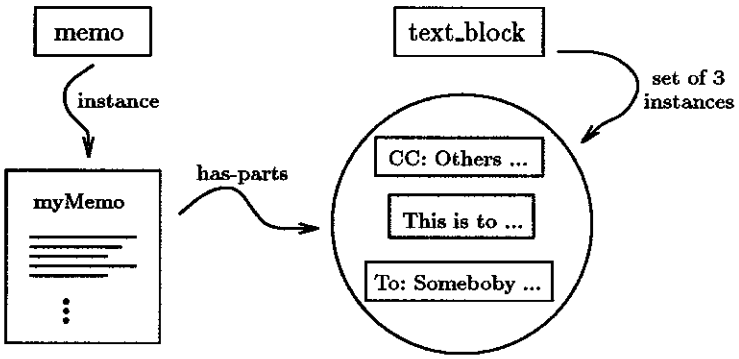

Fig. 3. Inadequate representation of memo

Another special type of multi-valued relation, the *ordered part relation*, accommodates an ordering among parts. For example, a memo can be modeled as the composition of a header, body, and carbon copy (CC) section. These latter objects can be modeled as "text block" objects. Hence, it might seem possible to represent `memo` with a fixed-cardinality part relation of degree three to `text_block` (Fig. 3). But throwing components together in a set with cardinality three does not preserve the ordering among them. In fact, there is no way to tell the header from the body, etc. Our ordered part relation resolves this problem by allowing an ordering to be specified among the parts.

An *essential* part relation requires the holonym to have exactly one part of the given type. It is therefore equivalent to the fixed-cardinality relation of degree one. An example is the relation between a car and its frame.

Dependency semantics are often desired when modeling with parts. If a large integral object, such as a CAD drawing, is deleted, then we may want to have all its parts deleted, so that these need not be searched out and deleted separately. Our model provides a *"part-to-whole" dependent* part relation for this purpose.

There are also times when a *"whole-to-part" dependency* is desired. Referring back to our music publication system, consider what happens when a particular instrument is removed from the orchestration of a score, meaning that the staff containing the instrument's music is removed from the ensemble score. In such a case, the staff object is deleted from the database. Since an instrument score consists of only a single staff, and since the staff contains the music, the deletion of its staff leaves an instrument score with no music at all. Therefore, the instrument score should be deleted as well. So it is sensible to make `instrument_score` dependent on `staff`. Our model provides this complementary "whole-to-part" dependent part relation.

In our part model, we define a general mechanism for value propagation [25] from both the meronym to the holonym, and vice versa. Value propagation refers to the flow of a data value across the part connection. As a modeling tool, it is useful for

expressing certain functional dependencies between integral objects and their parts. As an example, the attribute *age* of a car can be taken to be identically equal to the age of its frame. Hence, the value of *age* should be propagated upward through the part link from the `frame` to `car`. Such an arrangement not only alleviates the need to explicitly store *age* with `car`, it also eliminates the burden of having to maintain the functional dependency. By propagating the attribute's value, we insure that it is the same at both `car` and `frame`.

The direction of flow may also be from the whole to the part. Such a scheme captures the case where a data value of the whole determines something about its parts. For example, if a filing cabinet is composed of steel, then its drawers are probably composed of steel, too. In general, we could opt to model drawers such that they are always composed of the same material as their filing cabinets. We stress that within our part model, such an arrangement would not represent a default (see, e.g., [27]) and would not be defeasible, but would represent a definitive modeling decision. All drawers would be required to obtain their material make-up from their filing cabinets.

4 Realization of the Part Relations

In this section, we discuss our realization of the above mentioned part relations. Before getting to the specifics, we introduce our basic approach which is to represent the part relation as a type of object class—objects, in this case, representing relationships between other objects [1, 8, 23, 28].

4.1 Generic Part Relation

Assume that we have two classes, `A` and `B`, and we wish to define a part relation between them such that `B` is the meronymic class and `A` is the holonymic class. The syntax for skeletal definitions of the two classes can be seen in Fig. 4, where the line in the definition of `A` containing the keyword `has-part` indicates the part relation to `B`. Included in the `has-part` specification is an optional selector "myB" which can used by an instance of `A` to reference its part. The selector may be omitted, in which case the name of the meronymic class is used by default. To improve readability, an optional corresponding `is-part-of` can be placed in the meronymic class, as we have done for `B`. Normally, this is not needed and just introduces unnecessary coupling between the definitions. However, there are two types of part relations (discussed below) which do require this explicit reference to the holonymic class.

To realize the part relation between the two classes, our system expands the `has-part` construct into a subschema of its own. The system automatically defines an object class `B-PART-A` and connects it via the relationships *holonym* and *meronym* to `A` and `B`, respectively. The system also adds relationships *p1* to `A` and *w1* to `B`. The names *p1* and *w1* ("p" for part, and "w" for whole) are chosen arbitrarily by the system so as not to conflict with the names of any other properties of the class. As mentioned above, a selector (e.g., "myB" of `A` in Fig. 4) is used to retrieve the part from within the integral object. This "selector" is actually the name of a path method [14] defined by the system as the final step in the subschema expansion and

```
class A
     has-part(myB:B)

class B
     is-part-of(A)
```

Fig. 4. Incomplete definitions of A and B

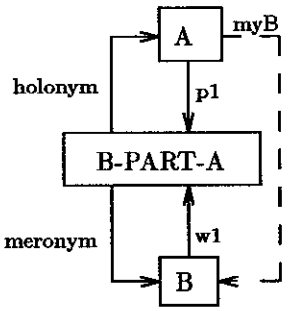

Fig. 5. Schema expansion for Fig. 4

added to the definition of the holonymic class. The method is defined such that it traverses *p1* to the relation object (i.e., an instance of B-PART-A) and returns the part object's OID which it obtains as the value of *meronym*. The entire subschema expansion for the part relation between A and B is shown pictorially in Fig. 5.

This realization was chosen for a number of reasons. First, the arrangement does not require the introduction of any extraordinary new OODB modeling constructs. In fact, all the different part relations can be obtained by making straightforward modifications to the basic configuration. The arrangement also permits traversal from the whole to the part, and vice versa. Furthermore, the mechanism offers a convenient way of performing value propagation along the part hierarchy. Finally, as an object in its own right, the relation can be endowed with attributes in a similar fashion to relationships in the ER and other semantic models.

4.2 Exclusive and Shared Part Relations

The part relation between **instrument_score** and **staff**, introduced above, will be used to demonstrate the realization of the *intra-class exclusive* link between an integral object and its part. The (partial) definitions of two classes, **instrument_score** and **score**, from the music publication database are shown in Fig. 6. Note that the **has-part** specification in **instrument_score** has been changed to **has-intra-excl-part**.

The subschema expansion for this part relation (Fig. 7) is actually the same as the one presented above for the generic example. The key point here is that the relationship *w1* is defined by the system to be single-valued. Because of this, an instance of **staff** may only be related to a single instance of **staff-PART-instrument_score**. This, in turn, means that it can be part of only one **instrument_score**. Hence, the particular instrument score has an exclusive hold on the staff with respect to other instrument scores. However, the configuration in no way precludes an instance of another class (e.g., an ensemble score), with its own part connection to **staff**, from making such a reference. Thus, the configuration indeed captures the desired intra-class exclusive part semantics.

Before introducing the *inter-class exclusive part relation*, we need to discuss the generic system operation **make-part-of** which is used to establish a part connection between a pair of objects. The operation takes two arguments, the OIDs of the inte-

```
class score
   attributes:
     title: titleType
     dedication: string
     year: yearType
     opus_number: integer
     pages: integer
   relationships:
     composed_by: composer
```

```
class instrument_score
     subclass-of(score)
     has-intra-excl-part(stf:staff)
     has-part(score_expression_sequence)
   attributes:
     instrument: instrumentType
```

Fig. 6. Some class definitions from music publication OODB

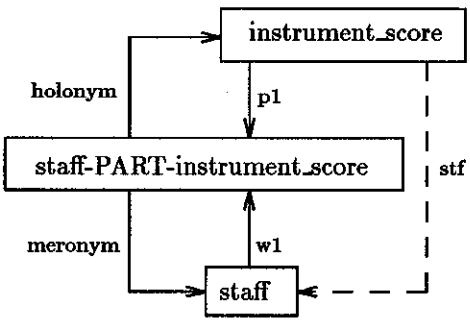

Fig. 7. The part relation between `instrument_score` and `staff`

gral object and the part. For instance, assume that we have an instrument score `SC` and a staff `ST`. Invoking `make-part-of(SC, ST)` causes the following actions to take place. First, an instance of the class `staff-PART-instrument_score` is created, and its relationships *holonym* and *meronym* are assigned values of `SC` and `ST`, respectively. After that, the relationships *p1* of object `SC` and *w1* of object `ST` are given the value of the OID of the new part relation object.

Because one inter-class exclusive part relation affects all the part relations of a meronymic class, it is not possible for the system to properly enforce the semantics of the restriction just by relying on the structure of the subschema expansion between the pair of classes. An operational approach is required. Our approach is to augment the writer methods [19] for the *wi*'s, the relationships automatically installed in the meronymic class by the system. The augmentation is such that the methods enforce the "make-component" rule defined in [21], as explained in the following.

Assume that we have a class `B` which is the meronymic class in n different part relations. The holonymic classes are `A1`, `A2`, ..., `An`. The system-defined relationships emanating from `B` to the part relation classes `B-PART-A1`, `B-PART-A2`, ..., `B-PART-An` are *w1*, *w2*, ..., *wn*, respectively (Fig. 8). The part relation between `A1` and `B` is taken to be inter-class exclusive.

Let z be the OID of the instance of `B-PART-A1` created when `make-part-of` is invoked with a pair of OIDs, a and b, representing instances of `A1` and `B`, respectively. Instead of just assigning *w1* the value z when it is invoked by `make-part-of`, the

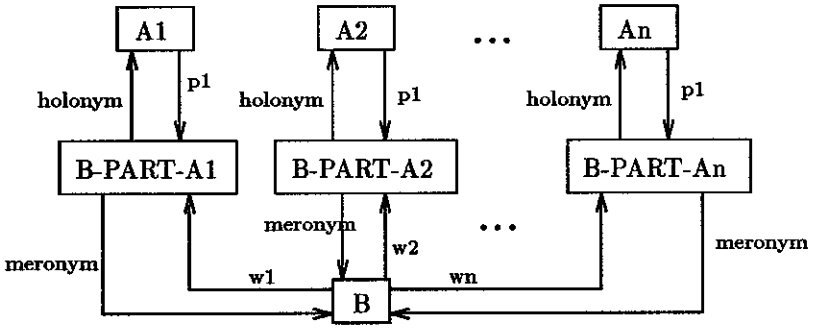

Fig. 8. A meronymic class with multiple part relations

writer method does the following. It scans the relationships *w2* through *wn* of the instance *b* searching for values other than nil. If it finds such a value, it knows that *b* is already part of an object other than *a* and thus cannot be grabbed exclusively by *a*. The writer then refuses to perform the assignment of *z* to *w1*. This failure implies the failure of **make-part-of** and guarantees the maintenance of the inter-class exclusive reference constraint. In the case of an intra-class exclusive or shared part relation, the writer method for the concomitant *wj* need only check the *wi*'s of inter-class exclusive relations for nil values; a meronym can participate freely in other intra-class exclusive or shared relations so long as it is not in any inter-class exclusive relation.

Syntactically, we represent the inter-class exclusive relation by replacing **has-part** with **has-inter-excl-part**. In Fig. 9, we see an inter-class exclusive part relation between the classes **car** and **engine**.

```
class car
     subclass-of(vehicle)
     has-inter-excl-part(engine)
   attributes:
     model: modelType

class engine
   attributes:
     model: engineModelType
     fuel: gasOrDiesel
```

Fig. 9. Classes car and engine

```
class instrument_score
     subclass-of(score)
     has-intra-excl-part(staff)
     has-shared-part
           (score_expression_sequence)
   attributes:
     instrument: instrumentType
```

Fig. 10. Instrument_score with shared part

A *shared part relation* is obtained in our model when the system-defined relationship *w1* is made multi-valued. By doing this, meronyms can be associated with many part relation objects, and thus with many integral objects. In Fig. 10, we show a revised version of the class **instrument_score** which indicates that many such objects can share the same score expression sequence. Note that **has-part** is replaced by **has-shared-part**. The schema expansion for this part relation is shown pictorially

in Fig. 11, where *w1* appears as a dual-lined arrow to indicate its multiplicity.

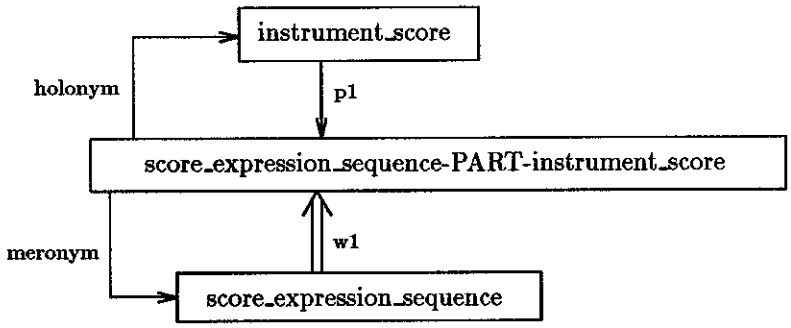

Fig. 11. Shared part relation of `instrument_score` and `score_expression_sequence`

4.3 Single- and Multi-valued Part Relations

Instrument scores, because they contain music for a single instrument, require only one staff. Hence the part relation between the classes `instrument_score` and `staff` is *single-valued*. In our realization, a single-valued part relation is obtained when the relationship *p1* of the meronymic class is defined to be single-valued (Fig. 7). This implies that an instance of `instrument_score` can be related to at most one instance of `staff-PART-instrument_score`, and consequently to only one staff.

On the other hand, ensemble scores are defined to have many staves, and so the part relation between the two respective classes is *multi-valued* (Fig. 12). The `has-part` construct, in this case, is modified with a pair of curly brackets surrounding its argument, conveying the fact that there are a set of parts from the given class. The schema expansion for the multi-valued part relation is obtained by defining *p1* to be multi-valued, as in Fig. 13. In this way, instances of the holonymic class can be related to any number of instances of the part relation class, and therefore any number of instances of the meronymic class.

```
class ensemble_score
      subclass-of(score)
      has-intra-excl-part({staff})
      has-intra-excl-part
          (score_expression_sequence)
```

Fig. 12. Multi-valued part relation

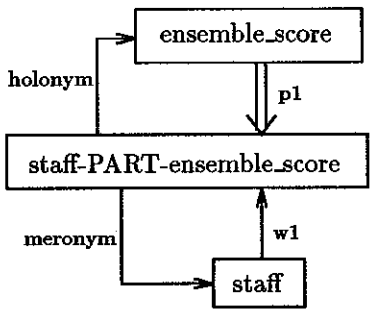

Fig. 13. Realization of Fig. 12

4.4 Range-restricted, Fixed-Cardinality, and Essential Part Relations

Above, we used the example of the engine with between four and twelve cylinders to define the *range-restricted part relation*. Here, we use the same example to discuss the realization. Syntactically, the range-restriction is expressed by adding a numerical range extension to the set notation as in Fig. 14. (The syntax used here is similar to Sowa's pluralization notation [30]). In terms of the subschema expansion, the range-restriction is obtained by placing upper and lower bounds on the cardinality of the relationship *p1* of the holonymic class [3, 4]. The subschema expansion for the part relation between the classes is shown graphically in Fig. 15. The numerical range 4–12 in parentheses following *p1* is our notation for the range constraint on this multi-valued relationship.

```
class engine
     has-inter-excl-part
          ({cylinder}:4-12)
   attributes:
     model: engineModelType
     fuel: gasOrDiesel
```

Fig. 14. Engines with 4 to 12 cylinders

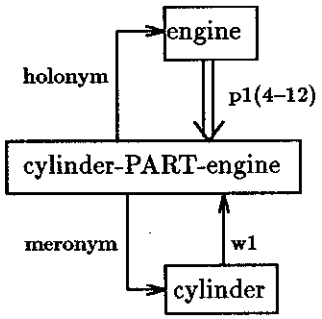

Fig. 15. Realization of Fig. 14

As the *fixed cardinality* and *essential* part relations are special cases of the range-restricted relation, their realizations are readily derived from the one just presented. Syntactically, however, we employ a different notation for each of these. For the fixed-cardinality relation, a single number is shown after the colon, instead of the numerical range (Fig. 16). The numerical range and curly brackets are omitted for an essential relation, and `has-part` is replaced by `has-essential-part` (Fig. 17).

```
class engine
     has-inter-excl-part
          ({cylinder}:6)
   attributes:
     model: engineModelType
     fuel: gasOrDiesel
```

Fig. 16. Engines with exactly 6 cylinders

```
class car
     subclass-of(vehicle)
     has-inter-excl-part(eng:engine)
     has-essential-part(frame)
   attributes:
     model: modelType
```

Fig. 17. `Car` with essential part `frame`

4.5 Ordered Part Relation

The class **memo**, discussed above, will be used to introduce the realization of the *ordered part relation*. The **has-part** specification for this relation includes a list of selectors in square brackets before the meronymic class's name (Fig. 18). Each element of the list is a selector for a single part object. (Cf. the generic part relation with the optional selector above.) To realize this part relation, the system places three relationships in **memo**, rather than one as with the other part relations discussed so far. It also creates three path methods to function as the selectors for the parts from within the holonyms. In general, the integral class must be equipped with n new relationships and methods. The schema expansion for the part relation between **memo** and **text_block** can be seen in Fig. 19.

```
class memo
  has-intra-excl-part
    ([header, body, CC]:text_block)
```

Fig. 18. Definition of class **memo**

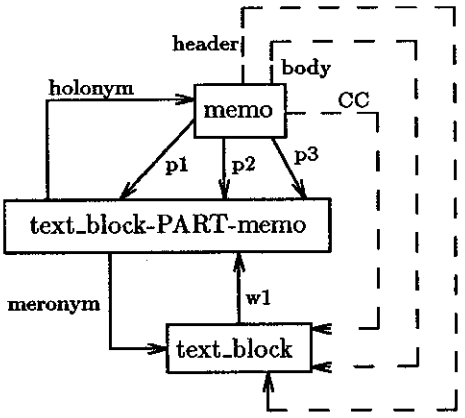

Fig. 19. Realization of Fig. 18

4.6 Dependent Part Relations

The realizations of the two kinds of *dependent part relations* can be derived by further refining the relationships *p1* and *w1* of the generic subschema expansion with dependency. As an example of the *"part-to-whole" dependent part relation*, assume that **block** is dependent on **engine**: If an engine is deleted from the database, its block is deleted, too. The definitions of these classes in Fig. 20 show that here, for the first time, an explicit reference (**is-dependent-part-of**) to the holonymic class is needed in the meronymic class. In the schema expansion in Fig. 21, we see that both *w1* and *holonym* are dependent (with dependency indicated by a double-headed arrow). Thus, if an instance of **engine** is deleted, the related instance of **block-PART-engine** is deleted, which in turn causes the deletion of the related **block**.

Referring back to our music publication system, we will use the part relation between the classes **instrument_score** and **staff** to demonstrate the realization of *whole-to-part* dependency. Remember that an instrument score contains a single

```
class engine
     has-inter-excl-part(block)
   attributes:
     model: engineModelType
     fuel: gasOrDiesel

class block
     is-dependent-part-of(engine)
```

Fig. 20. Engine and dependent part block

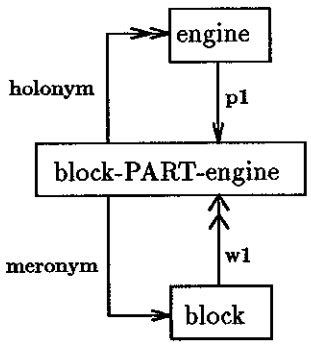

Fig. 21. Realization of Fig. 20

staff, and this staff contains all the music for the score. Without its staff, an instrument score contains no music at all. Thus, it is reasonable to make `instrument_score` dependent on its part `staff` (Fig. 22). To realize the dependency, the system designates *p1* and *meronym* dependent (Fig. 23), which causes deletions to be propagated from `staff` to `instrument_score` through `staff-PART-instrument_score`.

```
class instrument_score
   subclass-of(score)
   has-intra-excl-part-depends-on(staff)
   has-shared-part
        (score_expression_sequence)
  attributes:
   instrument: instrumentType
```

Fig. 22. `Instrument_score` with dependency

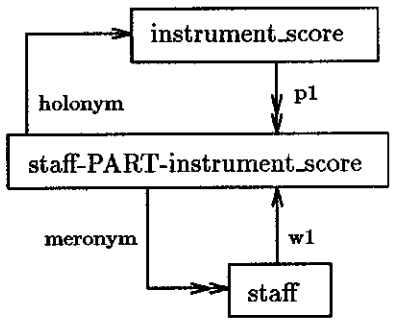

Fig. 23. Realization of Fig. 22

4.7 Part Relations with Value Propagation

The realization of upward value propagation is illustrated with the example of the propagation of *age* from `frame` to `car`. In Fig. 24, we show the textual definitions of these two classes, where the argument `frame` to the `has-part` statement is further qualified by the name of the attribute (*age*) whose value is to be propagated. In general, this secondary argument can be any method which is part of the meronymic class's public interface and which returns a data value.

To realize the propagation, the system installs, in the receiving class, a method which performs a traversal to the sending class and retrieves the value of interest (Fig. 25). In the figure, we see that the class `car` is augmented with the path method "age." The method operates as follows. It first crosses *p1* and arrives at the class

```
class car
      subclass-of(vehicle)
      has-inter-excl-part(eng:engine)
      has-essential-part(frame(age))
   attributes:
      model: modelType

class frame
   attributes:
      age: ageType
```

Fig. 24. `Car` with propagation of *age*

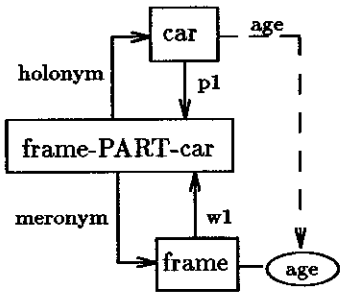

Fig. 25. Realization of Fig. 24

`frame-PART-car`. From there, it follows *meronym* to the class `frame` and accesses its namesake, the attribute *age* (through the selector method).

A similar scheme allows a value to be propagated down the part relation. Let's look at the example of the filing cabinet and its drawers (Fig. 26). Here, as in the case of part-to-whole dependency, an `is-part-of` statement is placed in the meronymic class. And as with `has-part` above, the argument to `is-part-of` is given an argument of its own (*material*), which is the property whose value is to be propagated. We once again emphasize that this value propagation is not a default: the value of a drawer's material is defined to be that of its filing cabinet.

```
class filing_cabinet
   has-inter-excl-part({drawer})
 attributes:
   material:
      one-of {steel, aluminum,...}

class drawer
   is-part-of
      (filing_cabinet(material))
```

Fig. 26. `Filing_cabinet` with propagation of *material*

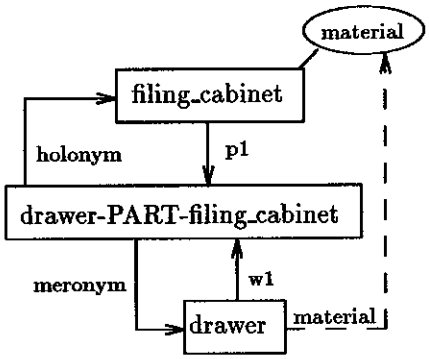

Fig. 27. Realization of Fig. 26

The realization of the downward propagation is analogous to that of upward propagation. Whereas for upward propagation, a method is installed by the system in the holonymic class, for downward propagation, a method is added to the meronymic class. In the example, a method "material" is installed in `drawer` to retrieve the value of the attribute *material* defined for `filing_cabinet` (Fig. 27).

5 Conclusion

In this paper, we have presented a comprehensive part model for OODB systems. Our model comprises a number of different part relations with characteristics such as exclusiveness/sharing, multi-valuedness, cardinality range-restriction, ordering, essentiality, dependency, and value propagation. All the relations were realized without modifying the underlying OODB data model. As a refinement of previous work, we have distinguished between two kinds of exclusiveness, intra-class and inter-class exclusiveness. The concept of dependency was refined to allow it in two directions, both from the part to the whole, and vice versa. We have also presented a general mechanism for upward and downward value propagation along the part relation.

In the tradition of the ER and other semantic data models, we have realized the part relation as a class whose instances represent the actual part connections between objects of the participating classes. Because of this, there is the possibility of defining attributes and even methods on the relation. In future work, we will consider how to exploit such capabilities. For example, the position of an integrated circuit on a circuit board might very well be stored as an attribute of the part relation between the respective classes.

An important issue is that of the transitivity of the part relation, which impacts on value propagation and "parts-explosion" retrieval. Ordinarily, transitivity is tacitly assumed. However, work in AI and related fields [7, 31] has shown that such an assumption is ill-founded. This issue is currently under investigation.

References

1. A. Albano, G. Ghelli, and R. Orsini. A relationship mechanism for a strongly typed object-oriented database programming language. In *Proc. VLDB '91*, pages 565–575, 1991.
2. J. Banerjee et al. Data model issues for object-oriented applications. In M. Stonebraker, editor, *Readings in Database Systems*, pages 445–456. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.
3. A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD Conference on Management of Data*, Portland, OR, May 1989.
4. R. J. Brachman. On the epistemological status of semantic networks. In N. V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 3–50. Academic Press, Inc., New York, NY, 1979.
5. R. Bretl et al. The GemStone data management system. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 283–308. ACM Press, New York, NY, 1989.
6. P. P.-S. Chen. The Entity-Relationship Model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
7. D. A. Cruse. On the transitivity of the part-whole relation. *Journal of Linguistics*, 15(1):29–38, 1979.
8. O. Diaz and P. M. Gray. Semantic-rich user-defined relationships as a main constructor in object-oriented databases. In *Proc. IFIP TC2 Conf. on Database Semantics*. North Holland, 1990.
9. D. Fischer et al. VML - The Vodak Data Modeling Language. Technical report, GMD-IPSI, Dec. 1989.

10. D. H. Fishman et al. Overview of the Iris DBMS. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 219–250. ACM Press, New York, NY, 1989.
11. J. Geller. *A Knowledge Representation Theory for Natural Language Graphics.* PhD thesis, SUNY Buffalo CS Department, 1988. Tech. Report 88-15.
12. J. Geller. A graphics-based analysis of part-whole relations. Research Report CIS-91-27, NJIT, 1991.
13. J. Geller. Propositional representation for graphical knowledge. *Int. J. Man-Machine Studies*, 34:97–131, 1991.
14. J. Geller, E. Neuhold, Y. Perl, and V. Turau. A theoretical underlying Dual Model for knowledge-based systems. In *Proc. of the First Int'l Conference on Systems Integration*, pages 96–103, Morristown, NJ, 1990.
15. J. Geller, Y. Perl, and E. Neuhold. Structure and semantics in OODB class specifications. *SIGMOD Record*, 20(4):40–43, Dec. 1991.
16. J. Geller and S. Shapiro. Graphical deep knowledge for intelligent machine drafting. In *Tenth Int'l Joint Conference on Artificial Intelligence*, San Mateo, CA, 1987. Morgan Kaufmann Publishers, Inc.
17. M. Halper, J. Geller, Y. Perl, and E. J. Neuhold. A graphical schema representation for object-oriented databases. In *IDS92, Int'l Workshop on Interfaces to Database Systems*, July 1992.
18. G. E. Hinton. Representing part-whole hierarchies in connectionist networks. In *Proceedings of the 10th Cog. Sci. Soc. Conference*, pages 48–54, 1988.
19. S. E. Keene. *Object-Oriented Programming in Common Lisp.* Addison-Wesley Publishing Co., Inc., Reading, MA, 1989.
20. W. Kim. A model of queries for object-oriented databases. In *Proceedings of the 15th Int'l Conference on Very Large Databases*, pages 423–432, 1989.
21. W. Kim, E. Bertino, and J. F. Garza. Composite objects revisited. In *Proceedings of the 1989 ACM SIGMOD Int'l Conference on the Management of Data*, pages 337–347, Portland, OR, June 1989.
22. B. MacKellar and F. Ozel. ArchObjects: Design codes as constraints in an object-oriented KBMS. In J. Gero, editor, *AI in Design '91.* Butterworth-Heinemann Ltd., 1991.
23. B. MacKellar and J. Peckham. Representing design objects in SORAC. To appear in *AI in Design '92*, 1992.
24. E. Neuhold, Y. Perl, J. Geller, and V. Turau. Separating structural and semantic elements in object-oriented knowledge bases. In *Proc. of the Advanced Database System Symposium*, pages 67–74, Kyoto, Japan, 1989.
25. G. T. Nguyen and D. Rieu. Representing design objects. In J. Gero, editor, *AI in Design '91.* Butterworth-Heinemann Ltd., 1991.
26. J. Peckham and F. Maryanski. Semantic data models. *ACM Comp. Surveys*, 20(3):153–189, Sept. 1988.
27. E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, Inc., New York, NY, second edition, 1991.
28. J. Rumbaugh. Relations as semantic constructs in an object-oriented language. In *Proc. OOPSLA '87*, pages 466–481, Oct. 1987.
29. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, NJ, 1991.
30. J. F. Sowa. *Conceptual Structures, Information Processing in Mind and Machine.* Addison-Wesley Publishing Co., Inc., Reading, MA, 1984.
31. M. E. Winston, R. Chaffin, and D. Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11(4):417–444, 1987.