

Value Propagation in Object-Oriented Database Part Hierarchies

Michael Halper, James Geller, and Yehoshua Perl
CIS Department and Center for Manufacturing Systems
New Jersey Institute of Technology
Newark, NJ 07102 USA

Abstract

Derived schema components are an important aspect of traditional semantic data modeling. In this paper, we address the issue of defining such schema constructs in the context of object-oriented database (OODB) part hierarchies. In particular, we present the concept of derived attribute defined with respect to value propagation across a part relationship between two object classes. Three different types of value propagation, namely, invariant, transformational, and cumulative, allow for a high degree of expressiveness in the definition of such derived attributes. We also present the notion of a generalized derived attribute, which may be defined in terms of simultaneous value propagations across many part relationships. The ambiguity problem of multiple value propagation in part hierarchies is solved by this latter construct, an analogue of which is not applicable in ordinary OODB IS-A hierarchies. It allows for the representation of such common expressions as the weight of the whole is the sum of the weights of the parts. To complement the formal definitions, we present a graphical schema notation for the value propagation mechanisms and the accompanying derived attributes. Such notation provides a convenient means for the specification and communication of OODB part schemata.

1 Introduction

Derived schema components have been a mainstay of traditional semantic data modeling [13, 23]. Their use tends to make a database's conceptual schema a more accurate reflection of the application domain it is designed to model. They also promote more concise representations and alleviate the burden of explicit integrity maintenance associated with redundant data storage [3].

Part hierarchies, where higher-level integral objects are constructed out of lower-level constituent objects, are a natural place in which to employ derived schema components. Often, a characteristic of a part is assimilated by its whole, or vice versa. For example, the color of a car can be defined as the color of its constituent body, or the font of a book's table of contents might be obtained from the book overall.

In previous papers [10, 11], we have introduced an extensive part model for object-oriented databases (OODBs) and presented a realization for it in the context of a general OODB data model. At the heart of this part model is a part relationship (following [16]) that relates a pair of object classes, and comprises constraints and functionalities which impose part-whole interaction on the instances of those classes. One of the characteristics of our part relationship is its value propagation mechanism, which forms the basis for the definition of derived schema components called *derived attributes*.

Value propagation refers to the flow of data values across the part relationship between parts and their wholes. It has been variously called attribute propagation [22] or local predication [27]. In our previous work [10], value propagation was limited to a single source object (either a whole or a part, depending on the direction of propagation), and data values were passed along "as is" without any intervening computation. This kind of propagation served to formalize the attribute propagation introduced in [22], and extended it by allowing propagation in either of the two directions across the part relationship.

In this paper, we enhance the value propagation mechanism in order to provide a more powerful means for defining derived attributes. In particular, we distinguish between three types of value propagation: *invariant*, *transformational*, and *cumulative*. The invariant version is that which we presented in [10]. Transformational propagation refines it by dropping the uniqueness requirement for the source of propagation and allowing for the specification of an additional computation during the propagation process. In this way, property values obtained from (possibly) many source objects can be transformed into a single value of a given data type. Cumulative propagation, a special case of transformational, collects the multiple property values into a set in a manner similar to union inheritance [29]. All three kinds of propagation also support a means for specifying defaults [24] in cases where there exists no source object or the desired property of that object is undefined. The *derivation* relationship of the SORAC model [19] also seems to lay the foundation for extensions to the attribute propagation of [22]; however, like [22], it informally introduces the construct and fails to distinguish properly among the different kinds of value propagation that occur in part hierarchies and the kinds of derived attributes that are induced by these processes.

In [10], we confined the definition of a derived attribute to a single part relationship, meaning that its value was

Appears in: B. Bhargava, T. Finin, and Y. Yesha, editors, *CIKM-93, Proceedings of the Second International Conference on Information and Knowledge Management*, pages 606–614, Washington, DC, Nov., 1993.

specified in terms of a property from a single class of parts.¹ However, it is often the case that derived attributes of the whole are best described in terms of identical properties from many of its parts, regardless of their classes. A canonical example of this is the weight of a car which is the sum of the weights of all its parts. Or, for example, the material make-up of a golf club is the set of materials from its shaft, head, and grip. To accommodate these situations, we extend our part model to allow for the definition of derived attributes in terms of simultaneous propagations of identical properties across many part relationships. In a pattern mirroring the part structure itself, the values of these multiple propagations are combined through a computation to form a value for the derived attribute. We will see that this mechanism serves as a natural third resolution strategy for the “multiple value propagation” (or “multiple inheritance” [28]) problem within part hierarchies. An analogue to this solution is not applicable in ordinary OODB IS-A hierarchies.

In [10], we introduced graphical notation for the part relationship in OODB schemata as a means for communicating about parts and facilitating part modeling. This notation enhanced a general graphical representation for OODB schemata which we introduced in [12]. In this paper, to complement the formal definitions, we present further enhancements to that notation. Specifically, we introduce graphical symbols for the three different types of value propagation and symbols for the various derived attributes defined with respect to these.

The rest of the paper is organized as follows. In Section 2, we review some of the terminology and notation that will be employed in the paper. In Section 3, we cover invariant value propagation. Section 4 introduces the notions of transformational and cumulative value propagation, while Section 5 covers generalized derived attributes, which are defined across many part relationships. Conclusions appear in Section 6.

2 Terminology and Notation

In this section, we present the terminology and notation which we will be employing throughout the paper. Following [30], we will often refer to a part object as a *meronym* (the prefix *mero-*, derived from the Greek *meros*, meaning “part”). A whole object will be called a *holonym* (the prefix *holo-* meaning “whole”). A part’s object class will be referred to as a meronymic class; that of a whole is a holonymic class. For example, if classes *engine* and *boat* are in a part-whole relationship, and the engine *e* is part of the boat *b*, then *e* is a meronym and *b* is a holonym. The classes *engine* and *boat* are the meronymic and holonymic classes, respectively.

The set $E(C)$ denotes the extension (i.e., the set of all instances) of the object class C . We will denote a part relationship between the meronymic class B and the holonymic class A as $P_{B,A}$. While in previous work [10] we have formally defined the part relationship $P_{B,A}$ as a multifaceted structure capturing a wide variety of semantics and functionalities, for the present purposes it suffices to view it as a relation (which we denote \diamond) from $E(B)$ to $E(A)$. The interpretation of $(b, a) \in \diamond$ is that instance *b* is part of instance *a*. We will ordinarily express $(b, a) \in \diamond$ in the infix form $b \diamond a$. In this context, we offer the following two definitions.

¹While both upward and downward value propagation are viable and our part model [10] addresses both, we will, without loss of generality, limit our discussion in this paper to the upward case, from part to whole, for the sake of brevity.



Figure 1: A part relationship between meronymic class B and holonymic class A

Definition 1: $\forall a \in E(A)$, let $M_{\diamond}(a) = \{b | b \in E(B) \wedge b \diamond a\}$. $M_{\diamond}(a)$ is called the *meronym set* of a with respect to the part relationship $P_{B,A}$. It is the set of instances of B which are parts of a .

Definition 2: $\forall b \in E(B)$, let $H_{\diamond}(b) = \{a | a \in E(A) \wedge b \diamond a\}$. $H_{\diamond}(b)$ is called the *holonym set* of b with respect to the part relationship $P_{B,A}$. It is the set of instances of A of which b is a part.

Following [31], we define the readable properties of a class [namely, its attributes, relationships, and (readable) methods] as functions which map the extension of the class into some given data type. For example, the attribute *height* of class *person* maps persons into the data type REAL. That is, $height : E(person) \rightarrow REAL$. A relationship, in this respect, poses no difficulty because it can be viewed as a special kind of attribute with type OIDType [4, 17] (i.e., an attribute which holds an object identifier). A property may be a partial function, i.e., it may be undefined for certain elements of its domain.

We will be using a graphical notation for describing object-oriented database schemata that we introduced in [10, 12]. In the following, we mention only the symbols which will be employed in this paper. For the rest of the graphical notation, see [10, 12]. A class is represented as a rectangle, while an attribute is represented by an ellipse attached to a class via a line. A (multivalued) relationship is denoted by a labeled (double) arrow directed from its source class to the referent class. A part relationship between the meronymic class B and the holonymic class A is represented graphically as in Figure 1. The symbol shown is employed for a number of reasons. First, the dashed line serves as a mnemonic device: The parts of the line indicate the parts of the whole. The fact that the line is bold makes the part relationship prominent in the overall context of the diagram; in this way, the hierarchy induced by the part relationship is clearly recognizable as a backbone of the graphical schema. The diamond head at one end of the line indicates the holonymic class. It is employed to maintain consistency with the OMT notation [25] and to avoid the impression of directedness, as the part relationship constitutes a powerful two-way access and constraint mechanism. The part relationship symbol serves as the basis for the new graphical notation that we will introduce.

3 Invariant Value Propagation

Often, when modeling with parts and wholes, a certain property of a part is naturally assimilated as a property of its

whole. For example, as we have mentioned above, it is sensible to define the color of a car to be the color of its constituent body [22]. Thus, we say that the attribute *color* of the class *car* is a derived attribute defined with respect to a value propagation across the part relationship between the classes *body* and *car*. In this example, it is obvious that the source of propagation, namely, a car’s body, is unique for each car. Furthermore, the value of the propagated property *color* at the class *car* is identical to the value of *color* at *body*. We refer to this kind of propagation as *invariant value propagation*. In general, a part relationship which performs such a propagation is defined as follows.

Definition 3: Let $\pi_B: E(B) \rightarrow \tau$ be a property of class B , where τ is some data type. The part relationship $P_{B,A}$ is *invariant-propagating with respect to property π_B* if it is single-valued or essential and it induces the property $\mathcal{D}_{\pi_B}: E(A) \rightarrow \tau$, called a derived attribute, on the class A such that:

$$\mathcal{D}_{\pi_B}(a) = \begin{cases} \pi_B(b), & \exists b \in M_{\diamond}(a) \wedge \pi_B(b) \text{ defined} \\ C, & \text{otherwise,} \end{cases}$$

where $C \in \tau$. We note that the derived attribute, the new property of A , is a function defined simply as the value of the property π_B for the part b when such a part exists for the given whole a . The condition that the relationship be single-valued or essential [10] guarantees the uniqueness of the source part b , ensuring that \mathcal{D}_{π_B} is well defined. In the case where the given whole a does not have a part $b \in E(B)$ or the part’s property π_B is undefined, then the derived attribute takes on the default value C [24], some constant value of the data type τ . This default value may be omitted with the consequence that the derived attribute may be undefined for some values of its domain.

For the example of the attribute *color* being propagated from *body* to *car*, the derived attribute \mathcal{D}_{color} is defined as:

$$\mathcal{D}_{color}(c) = \begin{cases} color(b), & \exists b \in M_{\diamond}(c) \wedge color(b) \text{ defined} \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

Here, we see that the color of a car is just the color of its body, if it has one. If it does not, then its color remains undefined.

Derived attributes, as with any derived schema components, should fit seamlessly into their respective classes and be accessible in the same manner as every other property. Toward this end, we augment the public interface of the class for which the derived attribute is defined with a message for reading it. Following the conventions discussed in [20], this message is taken to be identical to that for accessing the property at the meronymic class. So, in our example above, the class *car* would be given the additional message “color” which, for any instance, returns the value of the derived attribute \mathcal{D}_{color} .

To represent an invariant propagation graphically, a *propagation label* is written alongside the part relationship symbol. This propagation label comprises the name of the propagated property in parentheses and a preceding up-arrow, indicating the direction of the propagation.² The optional

²For downward propagation, from the whole to the part, a down-arrow is used instead of the up-arrow.

default value C , when utilized, follows the property name and is separated from it by a comma. The derived attribute, itself, is represented graphically as a dashed ellipse attached to the holonymic class via an unlabeled line. As we have discussed, the name of the derived attribute (i.e., its message in the public interface) is exactly the same as that of the propagated property. Thus, the same name which appears in the propagation label also appears as the name inside the dashed ellipse. Together, the propagation label and the dashed ellipse can be seen as symbolically defining both a derived attribute and its implementation in terms of the propagation of a data value across a single part relationship.

We note that, theoretically, the derived attribute symbol could be omitted and its existence inferred directly from the propagation label. That would be analogous to the representation of the IS-A relationship, where inherited values are not drawn at the subclass [12]. However, we have chosen to write the derived attribute explicitly for two reasons: (1) because it clarifies the schema without cluttering it; and (2) because it maintains consistency with the graphical representation of a generalized derived attribute (Section 5) which requires the ellipse. In regard to (1), cluttering is avoided because the propagation of properties in the context of the part relationship is selective, with only a small number of the holonymic class’s properties being defined in this manner. This, of course, is in contrast to IS-A, where all the properties of the superclass are inherited by the subclass.

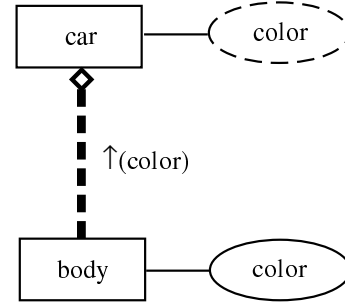


Figure 2: Color propagated from body to car

To illustrate the notation for invariant propagation, we show the example of the attribute *color* being propagated from *body* to *car* in Figure 2. Because the example did not include a default value, none appears in the propagation label.

4 Transformational and Cumulative Value Propagation

Because the value propagation mechanism is an instance-to-instance phenomenon, there is a potential for ambiguity regarding the source meronym when a holonym may have many parts of the same type. Note that there is no analogous phenomenon in normal IS-A hierarchies because even if objects’ representations are distributed up and down such a hierarchy between subclasses and superclasses [15], there is no chance that an object in a superclass will have more than one associated object in a given subclass. In the case of invariant value propagation, we explicitly eliminated the possibility of ambiguous sources by insisting that the part relationship be single-valued or essential (meaning that a holonym may have at most one part of the given type). In

this section, we generalize the value propagation mechanism by allowing more than one source meronym. As we will see, this potential ambiguity adds power and richness to the definition of derived attributes.

The way we deal with the (potentially) many source meronyms is to use algebraic tools. We transform the multiple values of the given data type provided by them into a single value with the use of a family of symmetric operators [5]. For this reason, we refer to this kind of propagation as *transformational value propagation*. As an example, in a document-archive system, the number of pages of an encyclopedia can be defined as the total number of pages in all its constituent volumes. As a special case of transformational propagation, we define *cumulative value propagation*, which is similar to the union inheritance found in some frame-based knowledge representation systems [29]. Instead of being transformed into a single value, the multiple property values are collected together into a set of the given type.

Now, let us formally define what we mean by part relationships which perform transformational and cumulative value propagation.

Definition 4: Let $\pi_B: E(B) \rightarrow \tau$ be a property of the class B , where τ is some data type. The part relationship $P_{B,A}$ is *transformational-propagating with respect to property π_B* if it induces the derived attribute $\mathcal{D}_{\pi_B}: E(A) \rightarrow \tau$ from the class A defined in terms of $\{T^{(n)}\}$, a family of symmetric operators $T^{(n)}: \tau^n \rightarrow \tau$, with $n > 0$, as follows. (Note that the meronym set of an instance $a \in E(A)$ is here taken to be $M\circ(a) = \{b_1, \dots, b_m\}$, $m \geq 0$, and $C \in \tau$.)

$$\mathcal{D}_{\pi_B}(a) = \begin{cases} T^{(m)}[\pi_B(b_1), \dots, \pi_B(b_m)], & m \neq 0 \wedge \\ & \pi_B(b_i) \text{ defined,} \\ & 1 \leq i \leq m \\ C, & \text{otherwise.} \end{cases}$$

In our example above, the derived attribute \mathcal{D}_{pages} which expresses the number of pages in an encyclopedia in terms of the number of pages in its volumes would be defined as:

$$\mathcal{D}_{pages}(y) = \begin{cases} \sum_{i=1}^n pages(v_i), & n \neq 0 \\ 0, & \text{otherwise,} \end{cases}$$

where v_1, \dots, v_n are the volumes of the encyclopedia y , and $pages: E(volume) \rightarrow \text{INTEGER}$ is the attribute of the class *volume* that holds a volume's number of pages. Here, we assume that *pages* is a total function, meaning that it is defined for all existing volumes. An encyclopedia without any volumes, by default, has no pages.

According to our definition, the property being propagated from the part to the whole is not restricted to a simple attribute but may be any of the readable properties of the class, including relationships and methods. Consider, for example, an authoring system used to write articles. Assume that there is a method *wordCount* defined on the class *section* which for a given section of text computes the number of words it contains. Using this method, we can define a propagating part relationship which computes the amount of words in an entire article. The derived attribute

$\mathcal{D}_{wordCount}$ defined on the class *article* is given as follows:

$$\mathcal{D}_{wordCount}(a) = \begin{cases} \sum_{i=1}^n wordCount(s_i), & n \neq 0 \\ 0, & \text{otherwise,} \end{cases}$$

where s_1, \dots, s_n are the sections of the given article a . As in the previous example, we assume that *wordCount* is a total function. An article is assumed to be of length 0 if it has no sections.

The third possible case is cumulative value propagation. Here, the induced derived attribute \mathcal{D}_{π_B} is set-valued, i.e., $\mathcal{D}_{\pi_B}: E(A) \rightarrow \{\tau\}$, where $\{\tau\}$ denotes a data type comprising sets of values of τ . It is defined as follows:

$$\mathcal{D}_{\pi_B}(a) = \begin{cases} \bigcup_{i=1}^n \{\pi_B(b_i)\}, & n \neq 0 \wedge \pi_B(b_j) \text{ defined,} \\ & 1 \leq j \leq n \\ C, & \text{otherwise.} \end{cases}$$

We see that this kind of propagation amounts to the union over the sets created through canonical injection of every property value into a singleton set. As usual, the default value is C which may be the empty set. As an example, let us return to the document-archive system and consider the part relationship between articles and compilations, the latter being collections of reprinted articles. To denote authorship, the class *article* has a relationship *writtenBy* directed to the class *person*. Using this relationship, we can define the overall authorship of a compilation through a cumulative value propagation. The derived attribute $\mathcal{D}_{writtenBy}$ of class *compilation* would be defined as:

$$\mathcal{D}_{writtenBy}(c) = \begin{cases} \bigcup_{i=1}^n \{writtenBy(a_i)\}, & n \neq 0 \wedge \\ & writtenBy(a_j) \\ & \text{defined,} \\ & 1 \leq j \leq n \\ \emptyset, & \text{otherwise,} \end{cases}$$

where c is a compilation and the a_i 's are its articles. It will be noted that, in this example, the propagated property *writtenBy* is a relationship, not an attribute.

The graphical schema notation for transformational value propagation is similar to that for invariant propagation, the difference being that the family of symmetric operators must be accounted for. To accomplish this, we modify the propagation label slightly. A symbol representing the entire family is written inside the parentheses in front of the name of the propagated property, which itself now appears in square brackets. Of course, in general, there will be no single symbol to denote the entire family. In such cases, we will employ a generic label (like "T") and place an annotation for it in a schema legend. For common families, such as summation, multiplication, min, max, and so on, which can be decomposed into closed, commutative, associative, binary operations, we use the ordinary operation symbol in the propagation label: E.g., upward propagating summation is written as $\uparrow(+[\pi])$, and multiplication is written as $\uparrow(*[\pi])$, where π is the propagated property. In Figure 3, we show how an encyclopedia derives its number of pages from its constituent volumes. Note that the part relationship symbol now comprises a dual-line indicating that, in general, encyclopedias have many volumes.

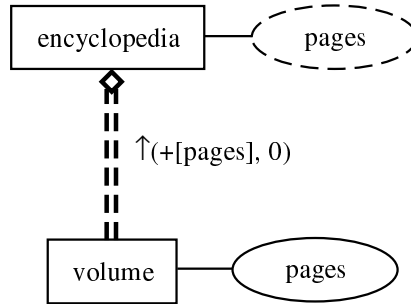


Figure 3: The number of pages of an encyclopedia as the sum of those of its volumes

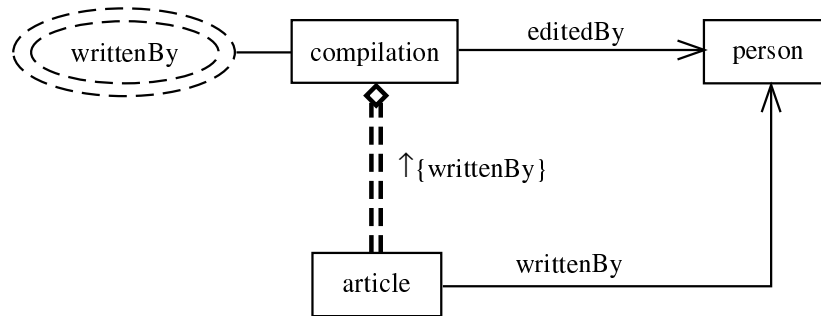


Figure 4: The authorship of a compilation determined from its constituent articles

We adopt a special convention for the propagation label of a cumulative value propagation. Instead of the parentheses, operator family symbol, and square brackets, the label simply comprises a pair of curly brackets enclosing the name of the desired property. The graphical representation of the derived attribute is also changed to a double ellipse to reflect the fact that it is set-valued. In Figure 4, we show the schema for the document archive containing compilations and their articles. Note that besides its derived attribute *writtenBy*, the class *compilation* has its own relationship *editedBy* with the class *person* used to denote editorship of a compilation.

5 Generalized Derived Attributes

In the previous sections, we introduced different ways to define derived attributes in terms of a single part relationship propagating the value of some property of the meronymic class. In this section, we consider the problem of defining a derived attribute with respect to many value propagations stemming from different part relationships simultaneously.

An obvious problem which can arise when specifying derived attributes in part hierarchies is similar to the so-called “multiple inheritance” problem in IS-A hierarchies [28]. In Figure 5, we illustrate the problem. There, we see multiple part relationships propagating the value of the same property π to a single class A . In this schema, it is not at all obvious what the value of π at A should be; in fact, the schema is not well defined.

To resolve this problem, we could employ one of the two strategies ordinarily used in IS-A hierarchies: Either we could disallow such ambiguous value propagation altogether and consider the schema invalid, or we could employ a prece-

dence list [14, 28]. However, in part hierarchies, there exists a third strategy which is a more natural solution. Often, it is sensible to model a property of the whole in terms of the values of the same property at its parts, regardless of their classes. Many examples readily come to mind: the color of an airplane is the combined colors of its fuselage, wings, nose, and tail; the weight of a car is the sum of the weights of its engine, drive train, frame, fenders, and so on; the materials of a golf club are those of its shaft, head, and grip; the reliability of a computer is the minimum of those of its monitor, CPU, disk drive, and keyboard; etc. We therefore view the inherent ambiguity of this “multiple value propagation” as a desired generalization of transformational value propagation across a single part relationship. We resolve it in an analogous manner by combining the multiple values with the use of a specified symmetric transformation. The new derived attribute induced by this process is called a *generalized derived attribute* because it is defined across many part relationships and supersedes those derived attributes which are induced by each part relationship individually. In a pattern mirroring the structure of the part hierarchy itself, the value propagation from each part relationship contributes to the value of the generalized derived attribute at the holonymic class. As before, the holonymic class’s public interface is augmented with a message (having the same name as the propagated property) to retrieve the value of this new property for a given whole. Let us now formally define what we mean by a generalized derived attribute and consider the conditions under which it is applicable.

Definition 5: Let π be a property (of data type τ) of each of the classes B_1, B_2, \dots, B_m (i.e., for all $1 \leq i \leq m$, $\pi: E(B_i) \rightarrow \tau$). Assume that we have part relationships

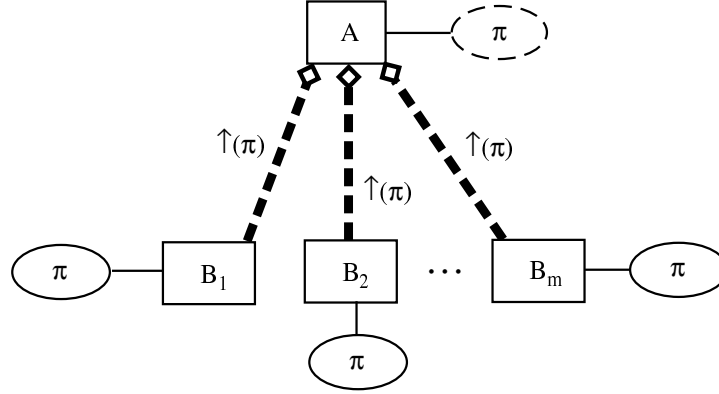


Figure 5: Redundant value propagation

$P_{B_1,A}, P_{B_2,A}, \dots, P_{B_m,A}$ such that each propagates the value of π up to A as in Figure 5. Assume also that there does not exist a part relationship $P_{A,Q}$ which propagates a value for π down to A . Furthermore, for all $1 \leq i \leq m$, let $\mathcal{D}_\pi^{(i)}: E(B_i) \rightarrow \tau$ be the function defined for $P_{B_i,A}$ which ordinarily serves as the definition for the derived attribute induced by that part relationship. The function $\mathcal{D}_\pi: E(A) \rightarrow \tau$, called a *generalized derived attribute*, is defined in terms of the symmetric operator $\psi: \tau^m \rightarrow \tau$ as follows (where $C \in \tau$):

$$\mathcal{D}_\pi(a) = \begin{cases} \psi[\mathcal{D}_\pi^{(1)}(a), \dots, \mathcal{D}_\pi^{(m)}(a)], & \mathcal{D}_\pi^{(i)}(a) \text{ defined,} \\ & 1 \leq i \leq m \\ C, & \text{otherwise.} \end{cases}$$

The generalized derived attribute resolves the redundant value propagations by combining the values of each individual propagation (i.e., the values of the functions $\mathcal{D}_\pi^{(i)}$, $1 \leq i \leq m$) into a single value through the symmetric transformation ψ . In this way, the value of the derived attribute \mathcal{D}_π for a given holonym $a \in E(A)$ is now determined by all a 's parts participating in a relationship propagating π , regardless of their classes. It should be noted that the definition makes no stipulation regarding the kind of value propagation that an individual part relationship may perform; it may be either invariant or transformational, with the function $\mathcal{D}_\pi^{(i)}$ defined accordingly (as in Definition 3 or Definition 4 above). Therefore, the generalized derived attribute may obtain a contribution for its value at some whole a invariantly from some lone part of a of a given type, or collectively from multiple parts through a transformation. The only requirement in this respect is that the resultant values from all the value propagations be of the same data type τ . We point out that if the provision excluding a downward propagation of π to A is violated, then the part hierarchy *in toto* is deemed invalid. As with the derived attributes from the previous sections, the generalized derived attribute may be given an optional default value C . If the default is omitted, then the generalized derived attribute may be undefined for certain elements of its domain.

As it stands, Definition 5 contains some restrictions that we adopted to simplify the presentation. These restrictions include the following:

1. The data type of the generalized derived attribute must be identical to the types of the propagated properties.

2. Cumulative propagation is not supported.
3. The data type of the property π at each meronymic class must be identical.

We will now consider how the relaxation of these restrictions generalizes our representation further.

As defined, the function \mathcal{D}_π yields a value of data type τ , the type of the propagated properties. In a manner analogous to cumulative value propagation across a single part relationship, we relax the first restriction by allowing the alternate form $\mathcal{D}_\pi: E(A) \rightarrow \{\tau\}$, where the generalized derived attribute is now defined to accumulate each part relationship's propagation value (taken as a singleton set) into a set of values of τ .

Regarding the second restriction, Definition 5 also assumes that all part relationships propagate a value of type τ (i.e., for all $1 \leq i \leq m$, the range of $\mathcal{D}_\pi^{(i)}$ is τ), the type of the property π at each meronymic class. As such, cumulative value propagation is not permitted in this context. To remove this restriction, we allow an alternative form of the operator ψ such that it takes arguments which are sets of values of type τ and, in turn, yields a single set of such values via some symmetric, set-theoretic operation. In this new form, $\psi: \{\tau\}^m \rightarrow \{\tau\}$. And, as in the case of cumulative propagation, the generalized derived attribute becomes set-valued: $\mathcal{D}_\pi: E(A) \rightarrow \{\tau\}$. We still do require that the type of value propagated by each part relationship, whether it is atomic or set-valued, be uniform across all relationships.

Although the property π at each meronymic class is taken to be semantically analogous to the same property at all other meronymic classes, the third restriction requiring that each have the same data type is often too limiting. For example, there may be discrepancies in the data types which really should not inhibit our ability to define a generalized derived attribute in terms of these properties. Consider the case where the weight of an airplane's engine is described in kilograms, while its fuselage's weight is given in pounds. Or consider the case where one of the weights is represented as an integer, and the other as a floating-point number. (One can find similar problems arising in the field of database integration [2, 26].) Because such discrepancies should not impede the definition of the generalized derived attribute, we adopt the following convention: As long as the data types of each π are compatible with each other, either in the sense that they have a common supertype in some type lattice [1] or that they can be cast into one another, then

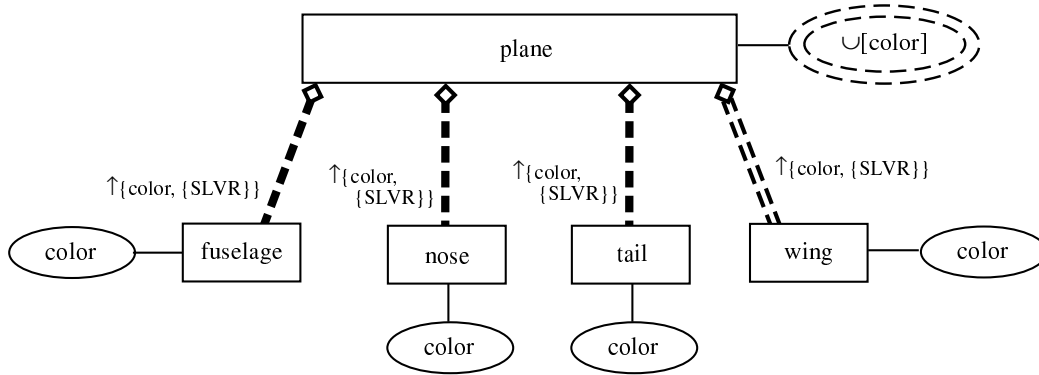


Figure 6: Plane getting its colors from its fuselage, wings, nose, and tail

the definition of the generalized derived attribute is deemed valid. However, we still insist that the types of the values propagated by each part relationship be identical, with any required type conversion incorporated into the family of operators $\{T^{(n)}\}$ defined by the schema designer for the individual part relationships.

To carry this point further, it may even be the case that the property π is set-valued at one meronymic class and single-valued in another, as when the fuselage of a plane is multi-colored, while its tail is of a single color. As above, if the values can be type-cast into uniform arguments for ψ (e.g., through a canonical injection of the atomic values into singleton sets), then the third restriction may be relaxed and the definition of the generalized derived attribute is admissible. Again, if one part relationship propagates a set, then all other relationships must do the same.

The graphical representation for a generalized derived attribute is based on the notation used for the derived attributes already presented. The only additional item is the operator ψ , which is handled in a similar manner to the family of operators in the propagation label of transformational value propagation. A symbol representing ψ is placed in front of the derived attribute's name, which is now bracketed, inside the dashed ellipse. Following our above stated convention, if the family of operators comprises related iterative binary operations, then the symbol for that binary operation (e.g., “+”) is employed in the schema representation; otherwise, some generic symbol and an annotation are required. The individual participating part relationships, as we have discussed, may be either invariant or transformational and retain their ordinary propagation labels to indicate their contributions to the computation of the generalized derived attribute's value. If the generalized derived attribute has a default value defined for it, then this is placed after its name inside the ellipse.

In Figure 6, we show how the color of a plane is defined as the union of the colors of its fuselage, wings, nose, and tail. The propagation from the class *wing* is a cumulative propagation because there may be several wing instances per plane. Because of this, the propagations from *fuselage*, *nose*, and *tail* require the transformation of the single color value into a singleton set. The derived attribute *color* of *plane*, being multivalued in general, is depicted by a double ellipse. The “ \cup ” in front of its bracketed name indicates that the operator family is that of the set-union operators. Therefore, the value of *color* (for a given instance of *plane*) is the union of the sets of colors propagated to it through

the four respective part relationships. Note that, by default, each part relationship sends a singleton set containing silver (written “SLVR”), the color of unpainted metal. Thus, there is no need to give the generalized derived attribute *color* at the class *plane* a default of its own.

To demonstrate that derived attributes may be manipulated in the same way as other class properties, let us consider a revised version of the above airplane schema where the description of fuselage is further refined into a set of constituent sections (Figure 7). Now, the property *color* of the class *fuselage* is itself a set-valued, derived attribute defined with respect to a cumulative value propagation from the class *fuselage_section*. Because of this, the propagation of *color* from *fuselage* to *plane* is no longer transformational but rather invariant as indicated by the ordinary parentheses enclosing “color” in the propagation label. The value propagation from *wing* remains cumulative, and, as before, the propagations from *nose* and *tail* must be cumulative in order to bring their data types in line with the other two.

In our final example (Figure 8), we show how a boat's weight may be written as the sum of the weights of its parts. Let us observe a few subtleties of this schema. First, the propagation label $\uparrow(+[\text{weight}], 0)$ of the part relationship between *engine* and *boat* indicates a transformational value propagation, whose contribution is a single weight value derived as the sum of the weights of all engines (of a given boat). In contrast, the similar expression $+[\text{weight}]$ appearing in the symbol for the generalized derived attribute at class *boat* means that the weight of a specific boat is the sum of the weights propagated to it from the classes *hull*, *engine*, *deckhouse*, and so forth. Note that the default value of each of the value propagations is 0. So, a boat without any parts has no weight.

6 Conclusions

In this paper, we have addressed the issue of derived schema components in the context of OODB part hierarchies. In particular, we introduced the concept of a derived attribute defined with respect to a value propagation across a part relationship connecting two object classes. Such derived attributes can be specified in terms of three types of value propagations. Invariant propagation passes along the value of a property from a unique source meronym to a holonym. The transferred value is delivered “as is” without any intervening computation. Transformational value propagation, on the other hand, relaxes the uniqueness requirement for

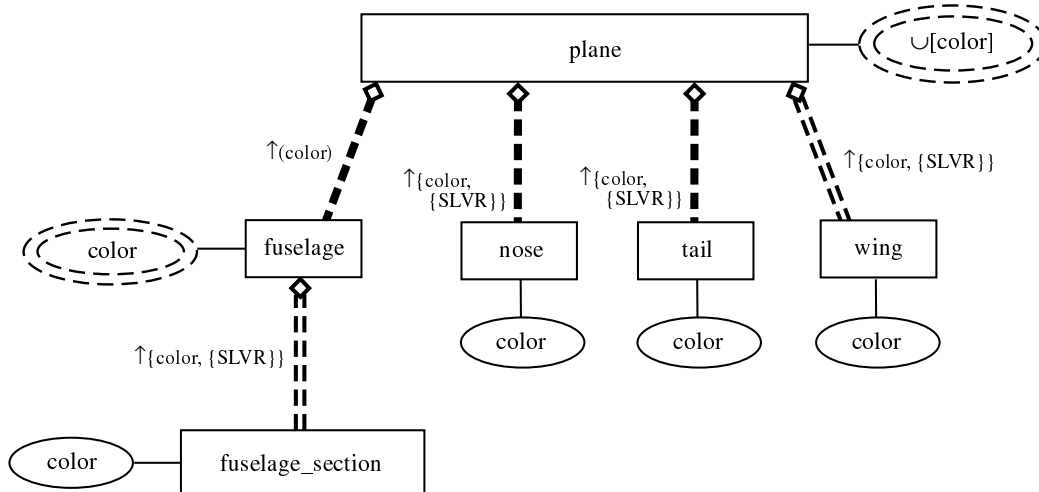


Figure 7: Fuselage getting its own color propagated from its constituent sections

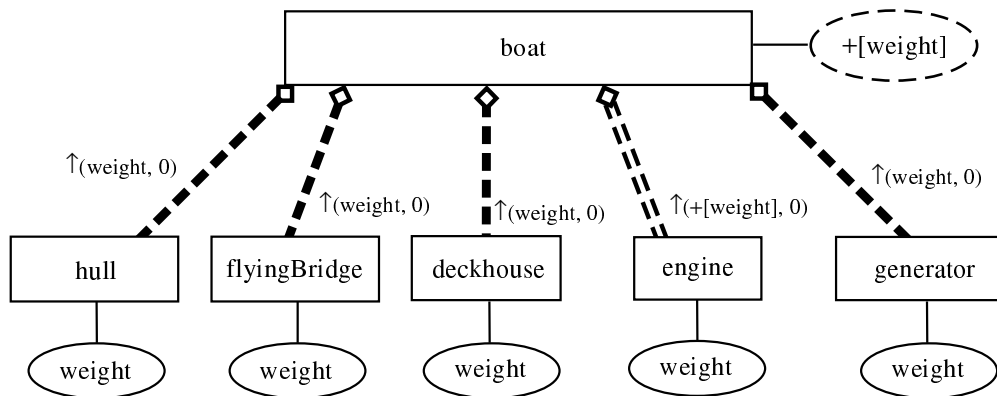


Figure 8: The weight of a boat as the sum of the weights of its parts

the source meronym and, using a family of symmetric operators, transforms the multiple property values into a single value of the specific data type. Cumulative value propagation is similar to transformational propagation, but instead of producing a single data value, it collects the multiple property values into a set which is propagated to the holonym.

To generalize these ideas, we also introduced generalized derived attributes which are defined with respect to value propagations across many part relationships. As we saw, they are a natural resolution strategy for multiple value propagation in OODB part hierarchies. They are also extremely useful for representing such common expressions as the weight of the whole is the sum of the weights of its parts, regardless of their classes.

To complement the formal definitions, we have presented a graphical representation for all the new schema constructs. This notation provides a convenient means for specifying and communicating about OODB part schemata. In particular, it allows for the symbolic representation of derived attributes and their implementations as value propagations across part relationships.

At present, we are in the midst of implementing our en-

tire part model in the context of Smalltalk [8]. This implementation will serve as a test-bed for experimentation and further research. We are also working on a realization of the part model in a real object-oriented database system, the VODAK Model Language (VML) of GMD-IPSI [4, 18] based on the Dual Model [6, 7, 21]. In this implementation, we are exploiting VML's open OODB model and its notion of metaclass [17] to realize the part relationship and all its various semantics, including value propagation and the accompanying derived attributes.

Acknowledgments

Thanks go out to Ken Sayers for his work on the Smalltalk realization. We would also like to thank Gisela Fischer and Wolfgang Klas of GMD-IPSI for their help with VML.

References

- [1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. The MIT Press, Cambridge, MA, 1985.

- [2] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–361, Dec. 1986.
- [3] C. J. Date. *An Introduction to Database Systems*, volume 1. Addison-Wesley Publishing Co., Inc., Reading, MA, fourth edition, 1986.
- [4] D. Fischer et al. VML - The Vodak Data Modeling Language. Technical report, GMD-IPSI, Dec. 1989.
- [5] J. B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley Publishing Co., Inc., Reading, MA, third edition, 1982.
- [6] J. Geller, E. Neuhold, Y. Perl, and V. Turau. A theoretical underlying Dual Model for knowledge-based systems. In *Proc. First Int'l Conf. on Systems Integration*, pages 96–103, Morristown, NJ, 1990.
- [7] J. Geller, Y. Perl, and E. Neuhold. Structure and semantics in OODB class specifications. *SIGMOD Record*, 20(4):40–43, Dec. 1991.
- [8] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley Publishing Co., Inc., Reading, MA, 1983.
- [9] R. Gupta and E. Horowitz, editors. *Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD*. Prentice Hall, Englewood Cliffs, NJ, 1991.
- [10] M. Halper, J. Geller, and Y. Perl. An OODB “part” relationship model. In Y. Yesha, editor, *Proc. 1st Int'l Conference on Information and Knowledge Management*, pages 602–611, Baltimore, MD, Nov. 1992.
- [11] M. Halper, J. Geller, and Y. Perl. “Part” relations for object-oriented databases. In G. Pernul and A. Tjoa, editors, *Proc. 11th Int'l Conference on the Entity-Relationship Approach*, pages 406–422, Karlsruhe, Germany, Oct. 1992.
- [12] M. Halper, J. Geller, Y. Perl, and E. J. Neuhold. A graphical schema representation for object-oriented databases. In R. Cooper, editor, *Interfaces to Database Systems*, pages 282–307. Springer-Verlag, London, 1993.
- [13] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Comput. Surv.*, 19(3):201–260, Sept. 1987.
- [14] S. E. Keene. *Object-Oriented Programming in Common Lisp*. Addison-Wesley Publishing Co., Inc., Reading, MA, 1989.
- [15] H.-J. Kim. Algorithmic and computational aspects of object-oriented database schema design. In [9], pages 26–61.
- [16] W. Kim, E. Bertino, and J. F. Garza. Composite objects revisited. In *Proc. 1989 ACM SIGMOD Int'l Conference on the Management of Data*, pages 337–347, Portland, OR, June 1989.
- [17] W. Klas. *A Metaclass System for Open Object-Oriented Data Models*. PhD thesis, Technical University of Vienna, January 1990.
- [18] W. Klas et al. Vodak design specification document. Technical report, GMD-IPSI, Nov. 1992.
- [19] B. K. MacKellar and J. Peckham. Representing design objects in SORAC: A data model with semantic objects, relationships and constraints. In *Second International Conference on Artificial Intelligence in Design*, Pittsburgh, PA, June 1992.
- [20] B. Meyer. Tools for the new culture: Lessons from the design of the Eiffel libraries. *Commun. ACM*, 33(9):68–88, Sept. 1990.
- [21] 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.
- [22] G. T. Nguyen and D. Rieu. Representing design objects. In J. Gero, editor, *AI in Design '91*. Butterworth-Heinemann Ltd., 1991.
- [23] J. Peckham and F. Maryanski. Semantic data models. *ACM Comput. Surv.*, 20(3):153–189, Sept. 1988.
- [24] E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill, Inc., New York, NY, second edition, 1991.
- [25] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.
- [26] A. P. Sheth and S. K. Gala. Attribute relationships: An impediment in automating schema integration. In *Workshop on Heterogeneous Database Systems*, Chicago, IL, 1989.
- [27] P. Simons. *Parts, A Study in Ontology*. Clarendon Press, Oxford, 1987.
- [28] A. Snyder. Encapsulation and inheritance in object-oriented programming languages. In *Proc. OOPSLA-86*, pages 38–45, 1986.
- [29] J. Walters and N. R. Nielsen. *Crafting Knowledge-Based Systems*. John Wiley & Sons, New York, NY, 1988.
- [30] M. E. Winston, R. Chaffin, and D. J. Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11(4):417–444, 1987.
- [31] S. B. Zdonik and D. Maier. Fundamentals of object-oriented databases. In S. B. Zdonik and D. Maier, editors, *Readings in Object-Oriented Database Systems*, pages 1–32. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.