Extending Aggregation Constructs in UML

Monika Saksena[1], Maria Larrondo-Petrie[2], Robert B. France[3], and Matthew P. Evett[2]

[1] PowerVision Corporation
220 Congress Park Drive, Suite 130
Delray Beach, FL 33445-4605, USA
Phone: (561) 279 2890
Fax: (561) 279 2810
msaksena@powervision.com

[2] Department of Computer Science & Engineering
Florida Atlantic University
Boca Raton, FL-33431-0991, USA
Phone: (561) 297 3855
Fax: (561) 297 2800
maria@cse.fau.edu

[3] Computer Science Department
Colorado State University
Fort Collins, CO 80523, USA.
Phone: (970) 491 6356
Fax: (970) 491 2466
france@cs.colostate.edu

Abstract. In this paper we provide a characterization of aggregation as used in static conceptual modeling of applications. Based on this characterization we suggest changes to UML notation that allow for more precise characterization of aggregate structures.

1 Introduction

Many authors in the software engineering, knowledge representation, conceptual modeling, and database communities have explored the notion of aggregation. For example, Winston et al. [7] characterized different kinds of aggregations from a knowledge representation perspective, Odell mapped this work to the OO modeling perspective [3], and Kilov and Ross [2] provide a rigorous treatment of aggregation in terms of invariant properties.

The *Methods Integration Research Group* (MIRG) is developing a characterization of aggregation that unifies and extends the above works on aggregation. Our initial focus is on aggregation constructs in static OO models (i.e., as used in Class Diagrams), and the results of our initial work are detailed in [6]. Our characterization of aggregation in static models is expressed in terms of primary (essential) and secondary (non-essential) characteristics. Combinations of the secondary properties result in varieties of aggregation. In this paper we propose extensions to the *Unified Modeling Language* (UML) [4] definition of aggregation that allow developers to explicitly express additional properties of the aggregate structures.

Fig. 1 shows our proposed classification of aggregation properties in terms of primary and secondary properties. The leaf nodes of the classification tree are properties. This characterization is independent of UML.

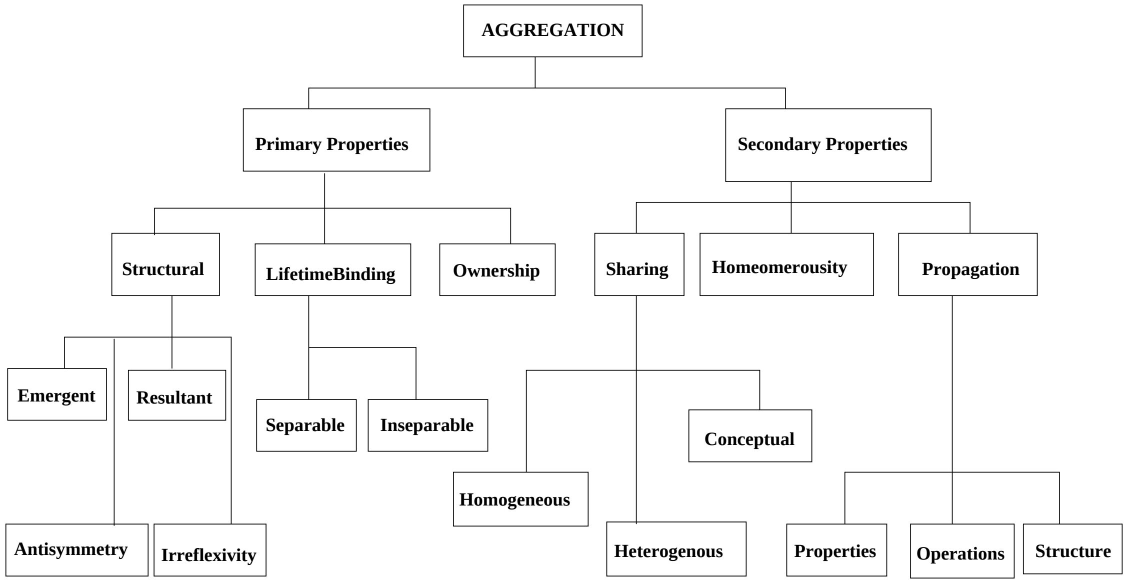

Fig. 1. Properties of Aggregation

2 Characteristics of Aggregation

In this section we give an overview of our proposed primary and secondary characteristics of aggregation. A primary characteristic is a property that holds for all forms of aggregation, while a secondary characteristic is neither unique nor essential to aggregation, but is used to distinguish different kinds of aggregation structures.

2.1 Primary Characteristics

The primary characteristics are grouped under three categories: *structural*, *lifetime binding*, and *ownership*.

Structural. The structural category includes static structural properties of aggregates. The mathematical properties required in an aggregate structure are antisymmetry and irreflexivity. We do not consider transitivity an essential property for all aggregations, for example, if a club is composed of persons and persons are composed of legs, then inferring that the legs of club members also belong to the club may not be desired. An aggregate structure must also have attributes based on *Resultant* and *Emergent properties* [2]. A resultant property of an aggregate structure is one that is dependent on a subset of the properties of the aggregate's parts. An emergent property is one that is independent of the properties of the component instances.

Lifetime Binding. Lifetime binding refers to the relationship between the lifetimes of the whole and its parts. The strong form of aggregation in the UML is defined as coincident lifetime binding between the part and the whole. The weaker form of aggregation relaxes this rule to allow sharing of parts across wholes. Civello [1] proposes that parts and wholes can outlive each other. In the Rumbaugh et al.[5] characterization of aggregation, the lifetime of the part is contained within the lifetime of the whole (i.e., parts do not outlive the whole). In our characterization of aggregation, we use the following weaker form of lifetime binding

> *In an aggregate structure, the lifetime of the part must overlap the lifetime of the whole.*

The possible relationships between lifetimes of parts and wholes allowed by our characterization are given in Fig. 2. Case 4 corresponds to the UML notion of coincident lifetimes. Cases 1 to 4 are scenarios of lifetime bindings where the lifetime of the part is contained within the lifetime of the whole. In these cases the parts cannot be separated from their wholes and must always be associated with a whole. We refer to this group of lifetime dependencies as inseparable parts in this paper. Cases 5 to 8 allow parts to exist without associated wholes. This group of dependencies is termed separable parts. In Cases 5 to 7 the death of a part is independent of the death of the whole. In Case 8, a part can exist on its own, but when associated with a whole, it dies at the time the whole dies.

Ownership. Ownership pertains to the control the whole has over the behavior of its parts. According to Odell, it is a term that describes the destiny of the whole with respect to its parts [3]. We consider the following notion of ownership as essential to aggregation:

> *An object owns its part in the sense that it controls the behavior of its parts, that is, the whole controls how and when the services of its parts are used. This requires that an aggregate class have methods that call the methods of the parts.*

Note that the above notion of ownership does not preclude other objects from interacting directly with the part. A part may be owned by one or more wholes at a particular time, and its owners may change over the part's lifetime.

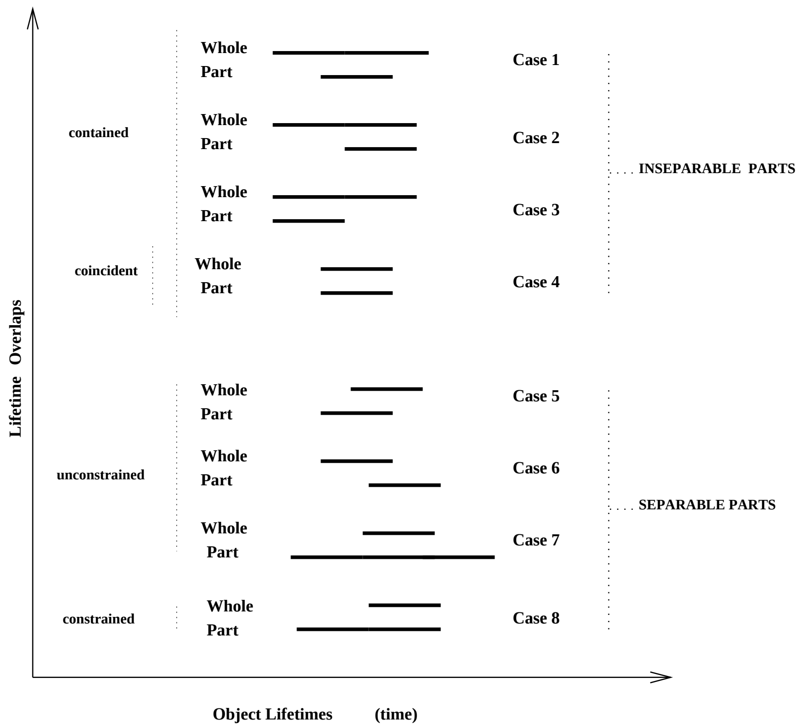

Fig. 2. Lifetime Bindings between the part and the whole

2.2 Secondary Characteristics

The secondary characteristics of aggregation are *parts sharing*, *homeomerousity* (similarity of part and whole), and *propagation of features*.

Sharing. Sharing of parts occurs when a part is simultaneously associated with more than one whole. Shared parts are necessarily separable. Three types of parts sharing can be identified:

- *Homogeneous Sharing*: An instance of a part class can be shared among different instances of the same aggregate class. For example, an instance of *Person* can belong to two different instances of *ResearchGroup*.
- *Heterogeneous Sharing*: An instance of a class can be shared by wholes that are instances of different classes. An instance of *Person* can be simultaneously shared by an instance of *Family* and by an instance of *ResearchGroup*.
- *Conceptual Sharing*: The concept represented by a class is shared. Conceptual sharing does not imply sharing of instances. For example, an *Engine* can be a part of two different aggregate classes: *Car* and *Plane*. However, one particular instance of *Engine* of an instance of a *Plane* cannot be used

as a part of an instance of *Car*. The concept of the *Engine* is being shared, not instances of *Engine*.

Homeomerousity. Parts in an aggregate structure are said to be homeomerous when they have properties in common with their whole or with other parts in the structure. For example, a homeomerous part class can be a specialization of its whole class. Such parts are often used to form recursive aggregate structures.

Propagation. Properties of the whole (attributes, operations, references) can propagate to its parts. Such propagation is a powerful and concise way of specifying an entire continuum of behavior. For an example of behavior propagation, consider a triangle modeled as an aggregate with edges as parts. Moving the triangle requires the movement of the edges. The move operation of the whole is propagated to its parts. For an example of attribute propagation consider a car composed of an engine, body and chassis. The color of the body is the color of the car; the color of the car propagates to the body.

3 Proposed extensions for UML

The information conveyed by existing graphical notations for aggregation is inadequate for representing various shades of aggregation. In this section we propose extensions to the standard UML notation to support a more precise modeling of aggregation characteristics. The primary structural properties of aggregation do not require any annotations because their semantics are implicit in the graphical representation.

While all aggregate structures have lifetime bindings, the precise nature of the bindings can vary across structures. The current UML notation does not directly support a more precise expression of the lifetime binding property of an aggregate. We propose that the parts in an aggregate be explicitly marked as being separable or inseparable. An inseparable component is marked by an *inseparable* annotation adjacent to the component class. For example, an inseparable *TableofContents* is depicted in Fig. 3.

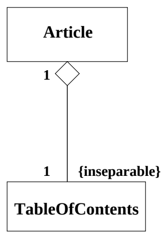

Fig. 3. Annotated Inseparable Part

A separable part of Property is shown in Fig. 4. In such cases, the cardinality of the whole class implies 0 as a lower limit for separable parts. This

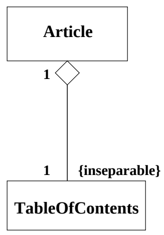

Fig. 4. Annotated Separable Part

notation provides for aggregate constructs in which one component is separable and another is inseparable.

The secondary properties require annotations because these connotations are not implicit in the definition of aggregation. The syntax for secondary properties is described next.

3.1 Sharing

Parts can be shared homogenously, heterogenously or conceptually. Homogenous sharing is depicted by multiplicities on the whole. For example, a *Person* object belonging to two *ResearchGroup* objects is indicated by placing a $0..*$ multiplicity at the *ResearchGroup* end of the aggregation. Heterogenous sharing is implicit when a class is part of two different aggregate structures. Conceptual Sharing is illustrated in Fig. 5, where the part class is annotated with *conceptual* in curly braces at the top of the component class.

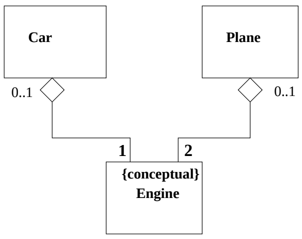

Fig. 5. ConceptualSharing

3.2 Homeomerousity

Homeomerousity is a special feature that is often applicable to certain cases where the part is similar to the whole. No annotation is required for homeomerousity, because it is evident in the definition and (recursive) structure of classes. An example is given in Fig. 6. An annotation is given in the example even though it is not needed.

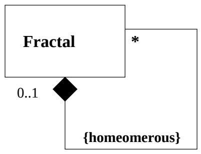

Fig. 6. Homeomerous parts and wholes

3.3 Propagation

Propagation of structure is implicit in the aggregate configuration. Operations and properties need special annotations since not all attributes and/or operations are propagated in any given structure. An arrow in the direction of propagation indicates propagation of the attribute to that class. For example, if the *color* of the *Body* is propagated to the aggregate class *Car*, then the attribute *color* has an arrow in the direction of propagation adjacent to it (Fig. 7).

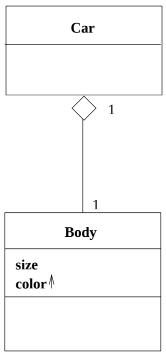

Fig. 7. Propagation of properties

Similarly, an arrow next to the affected operation can be used to depict propagation of operations. For example, Fig. 8 shows propagation of the operation, copy.

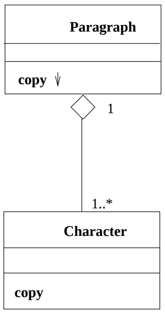

Fig. 8. Propagation of operations

4 Conclusions

We provide a characterization of aggregation that enhances the UML interpretation of aggregation. The characterization is expressed in terms of primary and secondary properties. Predefined UML annotations are proposed for rendering additional aggregation properties.

Future work will focus on providing a precise semantics for aggregation and identifying patterns of aggregate structures that are useful in conceptual modeling.

References

1. F.Civello. Roles for composite objects in object-oriented analysis and design. In *Proceedings of OOPSLA'93*, 1993.
2. H. Kilov and J. Ross. *Information Modeling: An Object-Oriented Approach.* Object-Oriented Series. Prentice Hall, 1994.
3. J. Odell. Six different kinds of composition. *Journal of Object-Oriented Programming*, 6(8), January 1994.
4. The UML Partners. Unified Modeling Language. Version 1.1, Rational Software Corporation, Santa Clara, CA-95051, USA, January 1997.
5. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.
6. Monika Saksena. The notion of aggregation. Master's thesis, Department of Computer Science & Engineering, Florida Atlantic University, Boca Raton, Florida, 1998.
7. M. E. Winston, R. Chaffin, and D. Hermann. A taxonomy of whole-part relations. *Cognitive Science*, 11, 1987.