A Characterization of Aggregation

Monika Saksena, Robert B. France, María M. Larrondo-Petrie

Department of Computer Science & Engineering
Florida Atlantic University
Boca Raton, FL-33431-0991, USA
{msaksena,robert,maria}@cse.fau.edu

Abstract. Most popular object-oriented modeling techniques (OOMTs) provide good support for the creation of structured conceptual models of system behavior and structure. A serious drawback of these techniques is that the concepts and notations used are loosely defined. This can lead to the creation of ambiguous models, and to disagreements over the proper use and interpretation of modeling constructs. An important modeling construct that is often loosely defined is aggregation. In this paper we present a characterization of aggregation that can help developers identify appropriate applications of the concept.

1 Introduction

There are several reasons for the popularity of graphical object-oriented (OO) modeling techniques (OOMTs) such as OMT [9] and Fusion [2]. They provide modeling concepts and constructs that allow developers to create conceptual models that closely reflect the real-world concerns that are pertinent to the system under development. These models can help developers gain insights into the essential behavior of systems.

A major drawback of popular OOMTs is that the notations and concepts they use are loosely defined. A result of this is that the models they produce can be ambiguous, and their interpretation often relies heavily on human intuition aided by the use of suggestive names and knowledge of the problem domain. Another serious consequence of poorly defined modeling concepts is that developers can waste considerable time resolving disagreements over usage and interpretation of the modeling constructs. For example, it is often not clear to developers when aggregation is to be preferred over general associations.

It is sometimes argued that loose interpretations are to be favoured because they allow the modeling concepts to be tailored to particular usages. This is indeed a strength if developers make clear the precise interpretation they used in creating their models. If this interpretation is not communicated to the reader of the model then the door is open for misinterpretation.

In this paper, we provide a characterization of *aggregation*, a modeling concept that is often the topic of heated discussions. In developing the characterization we examined published works on aggregation and analysed our OO modeling experiences. The characterization presented in this paper is the result of a careful analysis and distillation of these works and experiences, and is the first step towards formulating a formal semantics for aggregate structures. The ultimate goal of our research on aggregation is to provide a formal characterization that allows one to distinguish aggregation from the more general notion of association. The characterization presented in this paper is not presented in formal terms because it relects our initial understanding of properties pertinent to aggregate structures. As our understanding of aggregation properties deepens, a more formal characterization will evolve.

The characteristics we discuss are grouped into two classes: *primary* and *secondary* characteristics. Primary characteristics are properties that are essential to aggregation. They are invariant in the sense that they hold for all aggregate structures. Secondary characteristics are non-essential in the sense that they are not true for all aggregate structures. The secondary characteristics we discuss in this paper can also be found in other OO modeling constructs, however, in aggregation their manifestations can have special connotations.

In Section 2 we list the works that influenced our characterization. Section 3 presents the characteristics that we consider to be essential (the primary characteristics) and Section 4 presents some of the major secondary characteristics. We conclude in Section 5 with an outline of our planned work in this area.

2 Background and Related Works

In the OO community there is much debate on the usefulness of distinguishing aggregation as a special form of association. From the programming perspective the distinction between the two concepts is probably not useful because they can both be implemented in the same manner: as references to other objects (e.g., pointers to associated objects). We feel that a distinction can usefully be made at the conceptual modeling level. Modeling an association as an aggregation imparts additional information about the association. Unfortunately, this additional information is seldom clearly defined in published works on OO modeling techniques. This has led to confusion over the appropriate use of the aggregate construct, and has caused modelers to avoid using the construct.

The notion of aggregation has been explored by many authors in the software engineering, knowledge representation, conceptual modeling, and database communities. Winston *et al.* [11] characterize different kinds of aggregations from a knowledge representation perspective. In 1994, Odell mapped the work of Winston *et al.* to the OO modeling perspective [8]. Civello studied properties of whole-part associations in a case study [1], and Moreira and Clark present a rigorous treatment of aggregation [7]. Kilov and Ross [6] give a rigorous treatment of aggregation in terms of invariant properties.

This paper attempts to unify and extend the above works on aggregation. The Unified Modeling Language (UML)[4, 5], a recently approved Object Management Group OO notation standard, is used to represent the aggregation concepts discussed in this paper.

Our characterization of aggregation is expressed in terms of primary (essential) and secondary (non-essential) characteristics. Fig. 1 shows the organization of the characteristics discussed in this paper.

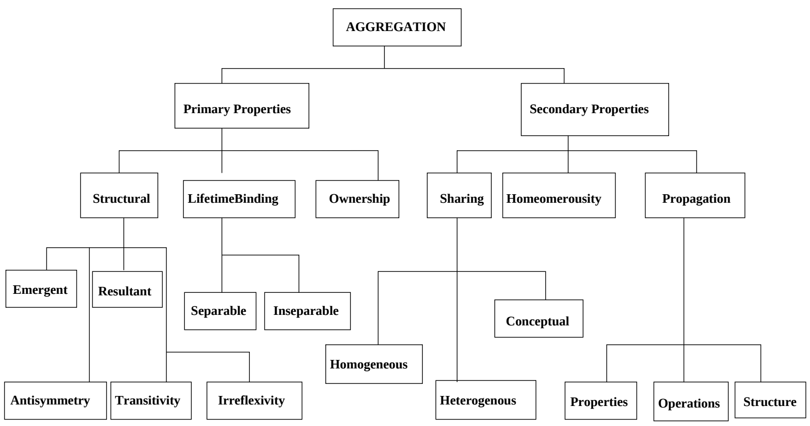

Fig. 1. Properties of Aggregation

3 Primary Characteristics of Aggregation

The primary characteristics are grouped under three views:

- **Structural:** This view focuses on the static structure of aggregates.
- **Lifetime Binding:** This view focuses on lifetime relationships between parts and their whole.
- **Ownership:** This view focuses on the behavioral relationship between the parts and their whole.

In the following subsections we elaborate on the characteristics in each of these views.

3.1 Structural Characteristics

Structural characteristics constrain the static aspects of aggregate structures. The transitivity, antisymmetry, and irreflexive characteristics are widely known and we simply state them below.
Transitivity

If object A is part of object B and object B is part of object C, then object A is part of object C.

Antisymmetry

> *If object A is part of object B then object B cannot be part of object A.*

Irreflexivity

> *Object A cannot be a part of itself either directly or indirectly. This means that the irreflexive property must hold for the transitive closure of the aggregate structure.*

The Resultant Property and the Emergent Property characteristics determine the types of properties (e.g., attributes) that an aggregate must have.

Resultant Property

> *A resultant property of an aggregate structure is one that is dependent on a subset of the properties of the aggregate's parts. It is dependent in the sense that a change in the properties in the subset results in a change to the resultant property. Every aggregate must have at least one resultant property [6].*

For example, in Fig. 2, *numberofsamples* is a property that is derived from the *Sample* parts.

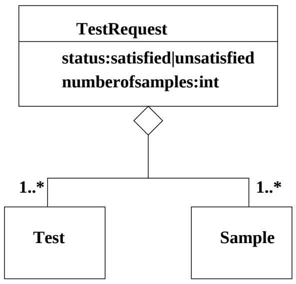

Fig. 2. Resultant and Emergent Properties

Emergent Property

> *An emergent property is one that is independent of the properties of the component instances. Every aggregate structure must have at least one emergent property [6].*

An example of an emergent property is the unique identifier associated with an aggregate instance. This property distinguishes an aggregate from its parts and justifies its existence. Emergent properties are independent of component properties, thus changing the properties of parts, or adding and deleting parts will not change the identity of an aggregate. An example of a non-identity emergent property is the *status* attribute of *TestRequest*.

It is important to note that resultant attributes of an aggregate class should not be modeled as attributes of the class if the relationships between it and its parts are modeled as associations rather than aggregations (see Fig. 3). The presence of resultant properties is a good indication that an association should be modeled as an aggregation.

Fig. 3. Alternative modeling as an association

3.2 Lifetime Binding Characteristics

Lifetime binding refers to the relationship between the lifetimes of the whole and the parts. In the Unified Modeling Language (UML) [5] the strong form of aggregation is defined with 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] puts forward the idea that parts and wholes can outlive each other. In the Rumbaugh *et al.* [10] characterization of aggregation, the lifetime of the part is "contained" within the lifetime of the whole (i.e., parts do not outlive the whole). Cook and Daniels [3] concur with this notion of containment.

In our characterization of aggregation, we use the weakest form of lifetime binding that seemed consistent with the notion of aggregation:

Lifetime Binding

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

All the possible relationships between lifetimes of parts and wholes allowed by our characterization are given in Fig. 4. Case 4 corresponds to the UML notion of coincident lifetimes. Cases 1-4 correspond to Rumbaugh's notion of contained lifetimes. Civello's notion of the part outliving the whole corresponds to Cases 6 and 7.

Cases 1 to 4 are scenarios of lifetime bindings where the lifetime of the part is described as "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. This group of lifetime dependencies is referred to as *inseparable parts*. Cases 5 to 8 allow parts to exist without associated wholes. This group of dependencies is referred to as *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 if it is associated with a whole, it dies when the whole dies.

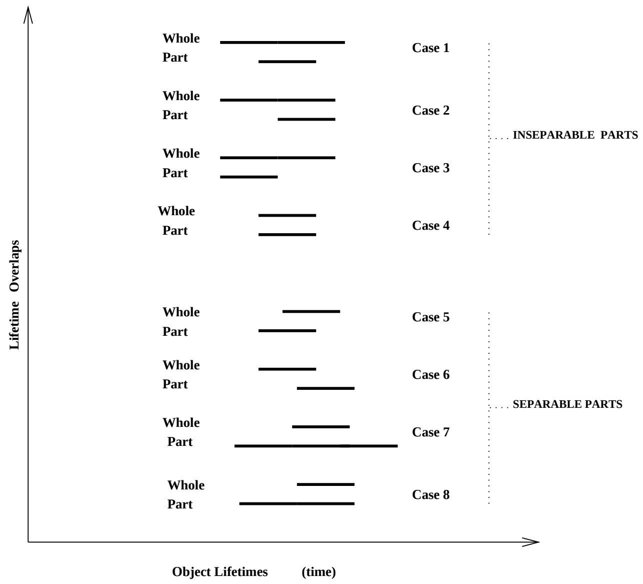

Fig. 4. Lifetime Bindings between the part and the whole

3.3 Ownership Characteristic

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 viz a viz its parts [8]. We consider the following notion of ownership as essential to aggregation:

Ownership

> *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. Ownership means that manipulation of the whole can involve manipulation of the parts. This notion of ownership facilitates the propagation of operations from the whole to its parts (see next section). A part may be owned by one or more wholes at a particular time, and its owners may change over the part's lifetime.

4 Secondary Characteristics

As stated earlier, secondary properties are non-essential properties of aggregation. Combinations of secondary characteristics yield different flavors of aggregation. The following classes of secondary characteristics are discussed in this section: parts sharing, homeomerousity (sameness of part and whole), and propagation of features.

4.1 Sharing of parts

Sharing of parts occurs when a part is associated with more than one whole at a particular time. Shared parts are necessarily separable.

The notion of sharing may seem alien to aggregation. Often, aggregation is based on the idea of "object containment", wherein parts are inseparable from their whole and can only be part of a single whole. In our characterization containment is not an essential characteristic of aggregation. 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, in Fig. 5 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. For example, in Fig. 6 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. Fig. 7 illustrates an *Engine* which is a class in two different aggregate classes: *Car* and *Plane*. However, an instance of *Engine* for 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*.

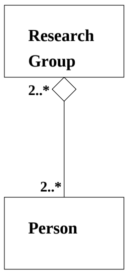

Fig. 5. Homogeneous sharing

Sharing implies that if the membership of the part in one aggregate is nullified then the part can continue to exist. *Constrained Sharing* occurs when a part exists only if it is a part of an aggregate structure (i.e., at any point in time when the structure is not being manipulated by an operation, the part must be associated with a whole).

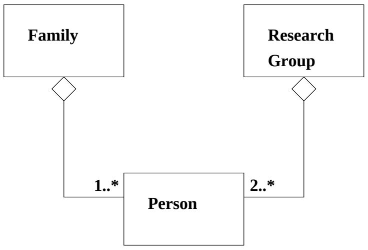

Fig. 6. Heterogeneous sharing

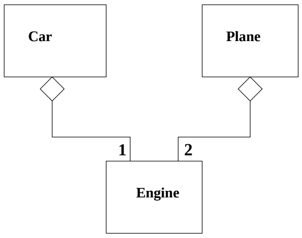

Fig. 7. Conceptual sharing

4.2 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.

4.3 Property 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 the Car composed of an $Engine$, $Body$ and $Chassis$. The color of the body is the color of the car, that is, the color of the car propagates to the body.

5 Conclusions

In this paper we propose a method independent characterization of aggregation. The characterization is expressed in terms of primary and secondary characteristics. This work is part of an ongoing effort to formalize OO modeling concepts. The objective is to create modeling notations that can be used to build precise and rigorously analyzable models. We are currently developing extensions of the UML that reflect the characteristics of aggregation outlined in this paper.

References

1. F. Civello. Roles for composite objects in object-oriented analysis and design. *OOPSLA*, pages 376–393, 1993.
2. Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes, and Paul Jeremaes. *Object-Oriented Development: The Fusion Method*. Prentice Hall, Englewood Cliffs, NJ, Object-Oriented Series edition, 1994.
3. S. Cook and J. Daniels. *Designing Object Systems Object Oriented Modelling with Syntropy*. Prentice Hall, Englewood Cliffs, NJ, 1994.
4. The UML Group. *Unified Modeling Language, Notation Guide, Ver 1.1*, August 1997.
5. The UML Group. *Unified Modeling Language, Semantic Guide, Ver 1.0*, January 1997.
6. H. Kilov and J. Ross. *Information Modeling An Object-Oriented Approach*. PTR Prentice Hall, Englewood Cilffs, New Jersey, 1994.
7. A. M. D. Moreira and R. G. Clark. Complex Objects: Aggregates. Technical Report TR-CSM-123, Department of Computing Science and Mathematics, University of Stirling, FK9 4LA, Scotland, August 1994.
8. J. Odell. Six Different Kinds of Composition. *Journal of Object Oriented Programming*, 6(8):10–15, January 1994.
9. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
10. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.
11. M. E. Winston, R. Chaffin, and D. Herrman. A Taxonomy of Part-Whole Relations. *Cognitive Science*, 11:417–444, 1987.