Aggregation from Multiple Perspectives by Roles*

Lars Kirkegaard Bækedal     Bent Bruun Kristensen
The Maersk Mc-Kinney Moller Institute for Production Technology
University of Southern Denmark/Odense University, DK-5230 Odense M, Denmark
e-mail: \{lkb, bbkristensen\}@mip.sdu.dk

Abstract

A whole is aggregated from parts, where a part can be aggregated itself or be atomic. The structure forms an aggregation (or part-whole) hierarchy. The whole can be modeled from multiple perspectives, where a perspective introduces its own aggregation hierarchy as well as dedicated properties for the whole and parts. Different perspectives share the same atomic parts. Role classes and objects support multiple perspectives on the aggregation of a whole with same atomic parts.

1: Introduction

From a collection of atomic parts we construct multiple aggregation hierarchies. The overall aggregated whole is the same for all hierarchies. The hierarchies model alternative perspectives on the aggregated whole. The atomic parts form the basis for all the hierarchies. The whole is described in different ways by different hierarchies. The actual structures of the hierarchies are typically different, and the properties of the whole, atomic parts and other elements can differ according to the perspective. The hierarchies must describe the same identity, but from different perspectives as reflected by properties and structure of the hierarchies. Figure 1 illustrates multiple aggregation hierarchies for the the whole $W$ and the atomic parts $P_1, P_2, \ldots, P_n$. Only two perspectives, $H_1$ and $H_2$, are illustrated.

![Figure 1. Aggregation from Multiple Perspectives](image)

Figure 1: Aggregation from Multiple Perspectives

For exemplification by a realistic example we look at an integrated software system for a shipyard [3], [4] as it contains a variety of principal problems, among others various perspectives on ships. Building large ships, e.g. super tankers and container carriers, involves design, manufacturing, economics, logistics, etc. Each of these tasks regards the ship from a particular point of view, a perspective. For a particular perspective we focus on structure and properties relevant to it. Figure 2 illustrates some of the perspectives which we find in the shipyard application.

---

*This research was supported in part by the Danish Academy of Technical Sciences, EF627.*
In section 2 we describe the use of roles to support multiple perspectives on the aggregation structures. In section 3 we focus on properties of wholes and parts in relation to multiple perspectives — disjoint and overlapping properties are supported differently. In section 4 we briefly summarize our approach to use roles for the support of modeling multiple perspectives in aggregation.

2: Roles and Multiple Aggregation Hierarchies

Usually a whole is a unique aggregation of its parts. By introducing perspectives we need an approach for on one hand to separate different aggregation hierarchies, on the other hand to actually relate these. This article proposes and illustrates an approach to obtain this based on role classes.

Figure 3 schematically exemplifies multiple perspectives of the same aggregated whole $W_1 = W_2$. The atomic parts are $P_1, \ldots, P_4$. Two hierarchies are illustrated by $A_{12}$ and $A_{34}$ for $W_1$, and $A_{13}$ and
The partial aggregates A12, A34, respectively A13 and A24, are different perspectives on the aggregation of P1, ..., P4 into W1 = W2 because the atomic parts are used differently.

**Roles.** Role classes support multiple perspectives on objects. In conceptual modeling [7] a role class is a language mechanism that models dynamic change of roles of an object. A role [8], [6] can be instantiated and the role object glued onto the intrinsic object (the object playing the role) and later removed from the object. Generalization and aggregation hierarchies model relations of role classes. Properties (instance variables and methods) of a role are additions to properties of the intrinsic object's class. In the description of the role class properties of the intrinsic class/object are directly accessible. An object and its currently available roles are seen as a subject with one identity. However, other objects can access this subject through its roles (by references that are typed by the role classes) to obtain different behavior from the subject.

![Figure 4. Role Classes: Illustration of Access](image)

In Figure 4 we illustrate access of role classes by means of references a, a1 and a2. We assume that class C and Ri qualifies a and a1, respectively. An object of class C can have one or several instances of role class Ri as its roles. Through the reference a of class C only method C.m is accessible — but not the Ri.ni methods. The reference a1 of role class R1 gives access to the methods R1.n1 and C.m — but not R2.n2. The implementation of method R1.n1 may invoke C.m, but the implementation of C.m cannot invoke R1.n1.

![Figure 5. Role Classes for Atomic Parts](image)

For each perspective we add a role to each atomic part. The role models properties needed to support a particular perspective for an atomic part. Properties of role classes and their support of perspectives are discussed further in section 3. In Figure 5 we illustrate two perspectives only, for the reason of simplicity. For the atomic part Pi we have added role classes Ri,1 and Ri,2.

**Multiple Aggregation Hierarchies.** For each perspective needed for an aggregated whole we form an aggregation hierarchy from roles of atomic parts instead of intrinsic atomic parts themselves. Role classes are aggregated into ordinary classes, from which the needed aggregation structure is organized as for ordinary aggregation. In Figure 5 we illustrate how the role classes Ri,1 and Rj,1 are aggregated into class A.

---

1C.m is a simple and yet precise notion for telling that we speak of method m belonging to class C.
Each resulting whole class for a given perspective is used in construction of a role class for the whole aggregate \( W \). In Figure 6 we illustrate such role classes for \( W \). To avoid confusion we have named the role classes \( RW_1 \) and \( RW_2 \).

Figure 7. Schematic Example with Aggregation Hierarchies

Figure 7(a) illustrates a partial solution to aggregation by a schematic example. It is similar to Figure 3. We have added (unnamed) roles corresponding to two different perspectives to the atomic parts \( P_1, \ldots, P_4 \). Two hierarchies are illustrated by \( A_{12} \) and \( A_{34} \) for \( W_1 \), and \( A_{13} \) and \( A_{24} \) for \( W_2 \). The intention is to make \( W_1 \) and \( W_2 \) the same whole class. The partial aggregates \( A_{12} \) and \( A_{34} \) form different perspectives on the aggregation of \( P_1, \ldots, P_4 \) by use of distinct roles of atomic parts. In Figure 7(b) we illustrate the rest of the solution where an aggregation hierarchy (an artificial or probably physical aggregation hierarchy) of \( W \) from \( P_1, \ldots, P_4 \) is included, and \( RW_1 \) and \( RW_2 \) are renamed as role classes of \( W \).

**Perspectives on Perspectives.** The resulting structure consists of a basic hierarchy and multiple perspective hierarchies. The root of each perspective hierarchy is a role class attached to the aggregated whole. Figure 8(a) depicts the overall structure. The solid line illustrates the basic hierarchy, whereas dotted lines illustrate perspective hierarchies.

Perspectives are also possible for a hierarchy that models a perspective itself. As illustrated in Figure 8(b) the hierarchy for the existing perspective also works as the basic hierarchy including its aggregated whole and its atomic parts. The hierarchy for the added perspective uses new role classes.
Figure 8. (a) Multiple Perspective Hierarchy. (b) Perspective on Perspective

which are added to existing role classes for atomic parts. The aggregated whole \( RW' \) becomes a role class for the existing aggregated whole \( RW'' \) (also a role class).

The Shipyard Example. During design activities naval architects, designers, engineers, draughtsmen and others first decide the general arrangement (overall aspects satisfying functional requirements to the ship) and later on the concrete implementation of it. They carry out a logical decomposition of the ship. The result is a complete specification of the steel structure (hull, decks, tanks, holds, bulkheads, girders, frames, webs, knees, plates, profiles, welding seam, etc.), outfitting (engines, winches, cranes, pumps, pipes, etc.) and electricity (cabling, light, instrumentation, etc.). To achieve their goal they use various simulations (e.g. sea keeping, manoeuvrability and impact) and evaluations (of stress, deformation, bending, buckling, fatigue, resonance, center of gravity, mass, hydrodynamics and many others).

Manufacturing impose another view of the ship. We are now interested in the assembly (composition) process rather than the decomposition process. Here, welders assembles plates, profiles, etc. into sections, sections into blocks and blocks into ships. Thus, manufacturing engineers break up the ship into a hierarchy of manageable components having no logical or functional meaning.

Logistics take care of “things to the right place at the right time”, in principle from ordering of materials to delivery of the ship. Essentially, logistics does not care much about the logical components of the ship. The manufacturing hierarchy, however, documents important information as sections and blocks make up units of elements that have a strong timely connection.

Economic analyses evaluate the price of various parts to ensure profitability. Both logical parts and assembly parts are subject to such analyses. Moreover, both estimated and actual price of each element are important to make budgets and find the final costs.

Even from this superficial introduction we observe at least 2 four alternative perspectives: design, manufacturing, logistic and economics. At least two structurally distinct aggregation hierarchies originating from design and manufacturing exist. Apparently, logistics imposes a hierarchy similar to the assembly hierarchy, but with different properties of components. Possibly, the economics perspective adapts to any hierarchy we might construct. That is, the economic perspective adapts the structure and attaches properties relevant to the economics. It would probably make sense to regard these as sub-perspectives. Figure 9 adds roles to the shipyard example. It is restricted to the hull and it also shows a possible physical perspective where a ship simply is a (huge) collection plates, profiles, pipes and welding seams.

\[^2\]It would possibly make sense to split the design perspective into several sub-perspectives. These could be perspectives for simulations, evaluations, compartments, structural system taking up loads, etc.
3: Roles and Properties

Each perspective introduces its own properties of a whole, of atomic parts as well as of remaining classes in the hierarchy corresponding to the perspective. We assume that methods and instance variables of a role class can support perspective-related properties. Sets of perspective-related properties may be disjoint or overlapping. In this article we use identical names to indicate overlapping properties. For independent (disjoint) properties the organization is straightforward, whereas overlapping properties cause further considerations.

Static and Dynamic Perspectives. A perspective and its aggregation hierarchy may be static or dynamic. In the dynamic case we can add a perspective to an aggregate with existing perspectives, and remove an existing perspective. The addition of a new perspective may interfere with existing perspectives in manners that are difficult to foresee. In the static case the perspective hierarchies are designed and constructed simultaneously such that their mutual relations are known. Thus an optimal, stable organization can be obtained because overlapping properties are known.

In Figure 10 the hierarchy of \( W_1 = W \) is used as the basic hierarchy. The partial whole \( RW_2 \) models a perspective on \( W \). We assume that the method \( RW_2.z \) is an independent property for the perspective \( RW_2 \). \( RW_2.z \) is modeled down through the hierarchy for \( RW_2 \) by means of ordinary invocation chains — not all invocations down through the hierarchy are shown. Invocation chains end at atomic parts — for example at the methods \( P2.y2 \) and \( P4.y4 \).

Overlapping Properties. Properties — instance variables or methods — of different perspectives can be overlapping. We shall assume that instance variables are accessible from outside an object only by means of access methods. We restrict ourselves from discussing the static case further. The properties are all known and we can group these optimally according to time and space efficiency \(^3\)

\(^3\)Time and space efficiency is in terms of the duplication of instance variables and methods.
and modeling accuracy \(^4\). A static hierarchy is just a dynamic hierarchy that never changes.

In the dynamic case we discuss properties of aggregated wholes. We assume that properties for an additional perspective overlap with properties of one or more existing perspectives. Also we exclude the special case where multiple perspectives can be seen as a perspective on one of the existing perspectives — else that is probably our solution. For the support of overlapping properties we choose between sharing or duplicating instance variables, and we use a mechanism to describe combined methods.

\[ \text{Figure 10. Example Illustrating Independent Property} \]

Figure 10. Example Illustrating Independent Property

Properties of Wholes. Assume that we want to add the RW2 hierarchy, and that two methods in RW1 and RW2 with identical names (x) must represent the same property of W (with role classes RW1 and RW2). We specify x as a subject method of W, RW1 and RW2 as illustrated in Figure 12. The intention is that all the bodies of these methods are invoked. The methods RW1.x and RW2.x, invoke the methods of the classes in the RW1, respectively RW2, hierarchy as illustrated in Figure 12.

We assume that overlapping instance variables in RW1 and RW2 are directly related to corresponding overlapping methods in the same classes. The instance variables are duplicated, and by

\[ \text{Figure 11. Combination of Properties of Role Classes} \]

Figure 11. Combination of Properties of Role Classes

\(^4\)That is, the naturalness of the model compared to our understanding of the problem domain.
combining corresponding methods we assure that instance variables cause no inconsistency problems. The overlapping — and duplicated — instance variables \(RW1.i\) and \(RW2.i\) are illustrated in Figure 12.

The method \(x\) may be a *reading* or an *updating* method (or both). A reading method will read some results from various hierarchies, possibly including the atomic parts, and combine partial results into a final result. An updating method will update elements in various hierarchies, including atomic parts. Atomic parts require special attention as they are shared among hierarchies — conflicting updates must be avoided.

**Properties of Atomic Parts.** Assume again that we want to add the \(RW2\) hierarchy and corresponding properties to atomic parts. Then, properties can be shared or duplicated. The subject method \(W.x\) can affect properties of atomic parts — its invocation can read and update properties of atomic parts. Inconsistent updates of duplicated properties and multiple updates of shared properties must be avoided. Intermediate parts of perspective hierarchies cause no similar problems, because these are assumed to be disjoint. For each atomic part overlapping properties are duplicated and several updates are necessary. This solution secures consistency on the expense of efficiency.

In Figure 13 we illustrate the \(y\) method and the instance variable \(k\) of the atomic part \(P\). The part \(P\) has the role classes \(R1\) and \(R2\) with methods \(y\) and instance variables \(k\). We assume that \(R2\) has been added as in the process of adding the perspective represented by \(RW2\). The methods \(y\) overlap, but they are not combined into one subject method — each individual method \(y\) is accessed by means of methods from its own aggregation hierarchy only. If the \(y\) methods had been combined into a subject method, then a reading method \(y\) would have caused no problems — it just reads from \(P, R1\) or \(R2\). However, an updating method \(y\) could cause problems — because as a subject method each of the \(y\)'s would be executed several times and possibly update the instance variables (k's) several times (also if only one \(k\) was available).

Figure 14 illustrates the organization for an overlapping property represented by the method \(x\). We assume that the method \(x\) is modeled down through both hierarchies — not all invocations are shown. Invocation chains end at atomic parts, e.g. at \(y\) methods of role classes of \(P2\). According
to our construction the \( x \) methods are combined to a subject method. However, the \( y \) methods are not combined into a subject method although they are used to model the overlapping property \( x \).

![Diagram of overlapping property](image)

**Figure 14. Example Illustrating Overlapping Property**

**The Shipyard Example.** For each perspective — design, manufacturing, logistics and economics — we identify relevant properties. If the property exists for more perspectives we also consider whether these are independent or overlapping. It should be noted, that a perspective may apply for a limited time. For instance, we would possibly discard a logistics perspective when delivering the ship as the information becomes useless. Perspectives should be more stable. For instance, we expect that the division of the design perspective into more fine-grained perspectives will show persistent as well as transient perspectives.

Figure 15 illustrates the model after addition of properties. It is not complete, and it only shows selected properties for two perspectives. Nevertheless it covers basic situations where a property is overlapping or independent (and both with and without name coincidence). The properties `design.move` and `manufacturing.move` constitute an example of a overlapping property with name coincidence. That is, moving a component in a design perspective also must affect the components position in a manufacturing perspective. The methods `design.move` and `manufacturing.move` in the roles of ship are combined as a subject method. The methods `design.move` and `manufacturing.move` in the roles of plates are not combined as a subject method, and the instance variable `position` is duplicated as `design.position` and `manufacturing.position`.

If the property `design.move` happened to be `design.translate`, it would be an example of a overlapping property without name coincidence. The properties `logistics.move` and `manufacturing.move` are independent properties with name coincidence. Figure 15 also shows a more complicated property where the name of a component is an aggregation of perspective-dependent names of parents, e.g. `ship.hull.plate`. This illustrates that a property of a part may depend on properties of its aggregation hierarchy.

**4: Summary**

In this section we briefly mention related and future work, and we characterize the approach presented in this article.
Figure 15. Properties in the Shipyards Example
Related Work. In [7] aggregation of the same whole from multiple perspectives is discussed, but not overlapping aggregation hierarchies or shared atomic parts. [1] (page 42) discusses abstraction from different perspectives, but not in terms of aggregation. At page 102 the aggregation relationship has two forms — a conceptual relationship is distinguished from physical containment. In [5] subjects model that different agents might view the same object from different perspectives. The agents not only have a filtered view of an object, but some of the methods of the object may be there only because of the perspective of an agent. Subjects do not involve multiple aggregation hierarchies.

The UML notation [2] distinguishes between simple aggregation which is entirely conceptual, and composition which introduces ownership and coincident lifetime. In simple aggregation, but not in composition, a part may aggregated into several wholes. The topic of this paper, identical wholes, identical atomic parts, and multiple aggregation hierarchies is not discussed.

Future Work. In general, the hierarchies of a multiple perspective aggregation model are disjoint, except for the atomic parts and the overall aggregated whole. An aggregated whole of this kind can itself be used as an atomic part of similar multiple perspective aggregation models. In general, any (subset of) an aggregation hierarchy can be replaced by a multiple perspective aggregation model. A potential problem is how the perspectives added to a class in its function as an aggregated whole match with the perspectives necessary for the class as an atomic part. As illustrated in Figure 16 the class W can have role classes as whole (W" with R4, R5, R6 and R7) and other role classes as an atomic part (W' with R1, R2 and R3). Because the perspectives are expected to be valid for the overall model, there will probably be no conflicts. Conflicts that are introduced due to dynamically changing model and perspectives can possibly be solved by simultaneously access of several roles [8].

![Figure 16. Match of Perspectives for Atomic Part = Whole](image)

Characterization. We characterize the approach to modeling of perspectives using multiple aggregates as proposed in this article as follows:

- There is always an implicit basic aggregation hierarchy which can be seen as physical aggregation. Any number of multiple aggregation hierarchies represent multiple perspectives on the aggregated whole can be seen as logical aggregation.
- Role classes added to atomic parts are aggregated into a ordinary classes. An aggregated whole for a perspective is a role class of the whole class.
- Various combinations of properties of role classes support requirements in the modeling of multiple perspectives in aggregation.
Acknowledgments. We thank Bob Basu, Kåre Groes Christiansen, Henning Kierkegaard, Kjell Harr, Nepper Larsen, Claus Risager, Gunner Schønfeldt, Carl Erik Skjelstrup at Odense Stell Shipyard for their contribution to the shipyard example.

References