A Conceptual Framework for Object Composition and Behavior Description

Dunia Ramazani and
Gregor v. Bochmann

Publication # 949

Département d'informatique et de recherche opérationnelle

Université de Montréal

Novembre 1994

A Conceptual Framework for Object Composition and Behavior Description

Dunia Ramazani
Gregor v. Bochmann

Département d'Informatique et Recherche Opérationnelle
Université de Montréal
C.P. 6128, succursale Centre-Ville
Montréal (Québec) H3C 3J7, CANADA

November 1994

Abstract:

A conceptual framework for object composition is outlined. It is based upon a simple abstract object model including the description of dynamic behavior. This model relies on ontological principles and recognizes three steps in the formation of composite objects: (1) configuration deals with the internal activity of the composite object; (2) juxtaposition determines the way the composite object is handled as a unit; and (3) emergence treats the new properties (properties which are not derivable or explainable by properties of component objects) which the composite object may acquire through the composition. The Ontological grounding renders the framework abstract and intuitive. It also allows the integration of existing approaches to composition. One of the major characteristics of this framework is a separation of concerns through the three steps of object composition .

Keywords: *Aggregation, Composition , Composite object, Decomposition, Framework, Is-part-Of, Nested objects, Object oriented development.*

This work was funded by the Ministry of Industry, Commerce, Science and Technology, Quebec, and the National Sciences and Engineering Research Council of Canada under the IGLOO project organized by the Centre de Recherche Informatique de Montreal.

1. Introduction

1.1. Three aspects of system structure

The object paradigm advocates a bottom-up model for application construction. This model uses three mechanisms to structure applications, namely subtyping (inheritance), composition (is-part-of, aggregation), and configuration (inter-object relationships). Generally, analyzing an application according to these three mechanisms leads to the identification of three aspects of an application, each serving a specific purpose:

(1) The aspect formed by focusing on composition and inter-object relationships shows the static and dynamic structure of objects of the application. It captures interactions among objects, and among objects and their component objects.

(2) The aspect formed by focusing on composition and subtyping relationships illustrates the static structure and categories of objects.

(3) Finally, the aspect consisting of subtyping and inter-object relationships shows categories of objects forming the application and their interactions.

This framework consisting of the three aspects for an application can serve to classify existing object-oriented analysis and design methods (OOADM) and programming languages (OOPL). In fact, three characteristics are illustrated by these aspects: the structure of objects in terms of other objects, the factorization of behavior among objects, and the inter-object behavior.

Existing OOADM and OOPL can be rated according to the coverage of these aspects. For instance, database applications tend to emphasize the composition and subtyping relationships, ignoring the inter-object behavior. As a matter of fact, OOADM well-suited for database applications will emphasize these aspects. On the other hand, OOPL are recognized to give little consideration to composition relationships. What is of importance to programming is how objects interact and whether they have some common behavior which needs to be factorized. In other domains, like Network management, subtyping relationships combined with composition relationships may suffice to describe the architecture and the behavior of a system. As coined by Bapat [Bapat 94], neither subtyping nor composition by itself is sufficient to describe the architecture and the behavior of a

management system; they are both required together. He motivates this necessity by the following analogy: "The subtype hierarchy[1] helps us to find the parts we need, because it has sorted and categorized them. Once we have found them, the composition hierarchy tells us what to do to put them together meaningfully. A subtype hierarchy is like a hardware store, i.e., it has aisles for nails, screws, brackets, two-by-fours, and so on. Without the sorting and categorization provided by the aisles (supertypes), shelves (subtypes) and bins (leaf types, i.e., types without subtypes), it would be difficult to find the parts we need. A composition hierarchy is like an assembly blueprint, i.e., once we have all the parts, it tells us how to put them together."

The above three aspects are interrelated. The complexity of an application may be strongly influenced by the interrelations among these aspects. We believe that in the specification of composite objects, there is such a complexity, particularly considering the behavioral interrelations among these three aspects, as this will be explained and exemplified in the sequel. It follows that adequate handling of these three aspects is a prerequisite for better support of composition of objects. Unfortunately, current object-oriented programming languages provide poor support for composition of objects [Johnson, Opdyke 93]. On the other hand, existing OOADM offer some kind of support for object composition.

1.2. Approaches to modelling composition

It is worth stating at this point that there is no standard meaning of composition in OOADM. We mention in particular the following approaches.

Entity/Relationship approach

Some methods express composition of objects as the abstraction of a given relationship among component objects. This is a reminescent of E/R methods. Most of the methods using an E/R basis, as the Fusion method [Coleman et al. 94], represent composition in this way.

[1] A composition hierarchy is the hierarchy made of types related by is-part-of relationships. The subtype hierarchy is the hierarchy of types related by subtyping relationships.

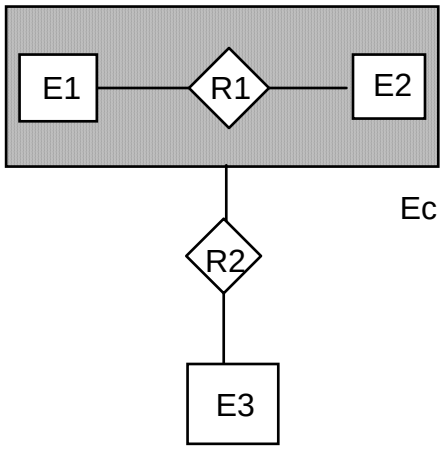


Figure 1: Composition as relationship abstraction

In Figure 1, Ec is a composite entity (object). E1, E2, and E3 are simple entities (objects). The relationships R1 relates E1 entities to E2 entities, and R2 relates E3 entities to Ec entities. Each Ec entity represents a pair <Ei, Ej> of E1 and E2 entities. To handle this pair <Ei, Ej>, the relationship R1 is abstracted to form a new entity Ec which attributes are those of the relationship R1. For instance, E1 may represent a set of students, E2 a course taken by these students and R1 the examination given by a professor represented by E3. If one has to find all the exams given by a certain professor, then it is convenient to think about exams as entities on their own. Thus, E1 and E2 may be aggregated to form a composite entity Ec which is the exam. An exam has certain attributes like its name, date, room and conditions related.

This way of handling composition shows the glue between the components objects. It focuses on the structure and neglects the behavior of the composite entity (object) formed. In this respect, it is more appropriate for database applications.

Is-part-of approach

Other methods focus on the relationship between the composite object and its component objects, the is-part-of (aggregation, composition) relationship. Database researchers found that we need to focus on is-part-of relationships in order to capture the semantics of complex physical assemblies [Liu 92].

In Figure 2, we illustrate such an approach through the description of a PBX. A PBX is made of cards which provide the functionality of the PBX such as trunk connections or

intercom calls. For each card, there is a number of ports which handle communication signals.

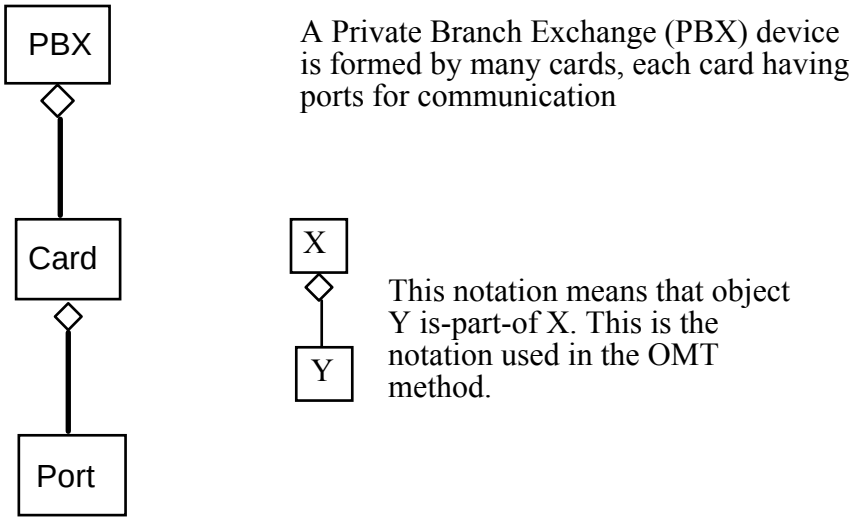


Figure 2: Composition reduced to is-part-of relationship

This approach focuses solely on the aspects of the is-part-of relationship like cardinality, exclusivity/sharedness, and dependency/independency of component objects which are explained later. However, like the Entity/Relationship approach, it neglects how the component object participates in the behavior of the composite object. It has the advantage of describing, in an hierarchical manner, the structure of an object.

Programmatic approach

Another approach is to view composition of objects as attribution, i.e., component objects are represented by attributes of the composite object. By expressing the previous example with this way of handling composition, we get the following result:

```
class PBX {                          class Card {                    class Port {
//other definitions...               //other definitions...          //other
Card PBX_Cards[n];                   Port Card_Ports[m];             //definitions//...
//n is the number of cards           //m is the number of ports      };
//of this pbx...                     // of this card ...
};                                   };
```

Figure 3: Programmatic approach of composition

Unexpectedly, this is the most popular way of handling composition of objects. This programmatic view of composition has the serious drawback of precluding any distinction between is-part-of (aggregation, composition) relationships and other associations among objects. We believe that such a distinction is necessary.

Multiple inheritance approach

There are also some situations where composition of objects is modeled by multiple inheritance. Such situations are reported in [Cargill 91, Rumbaugh 93, Sakkinen 89]. For example, in the Figure 4, an apple orchard is presented as a subtype (subclass) of both orchard and appletree. Rumbaugh [Rumbaugh 93] states that the correct way of representing such a situation is to model an apple orchard as a subtype of only orchard and this apple orchard will contains appletrees. Modeling composite objects by multiple inheritance is appropriate when the identity of component objects is not important and the focus is on the resulting properties of the composite object. This way of handling composition, while treating both structural and behavioral aspects of composition, can create serious problems for the reuse of the design[2].

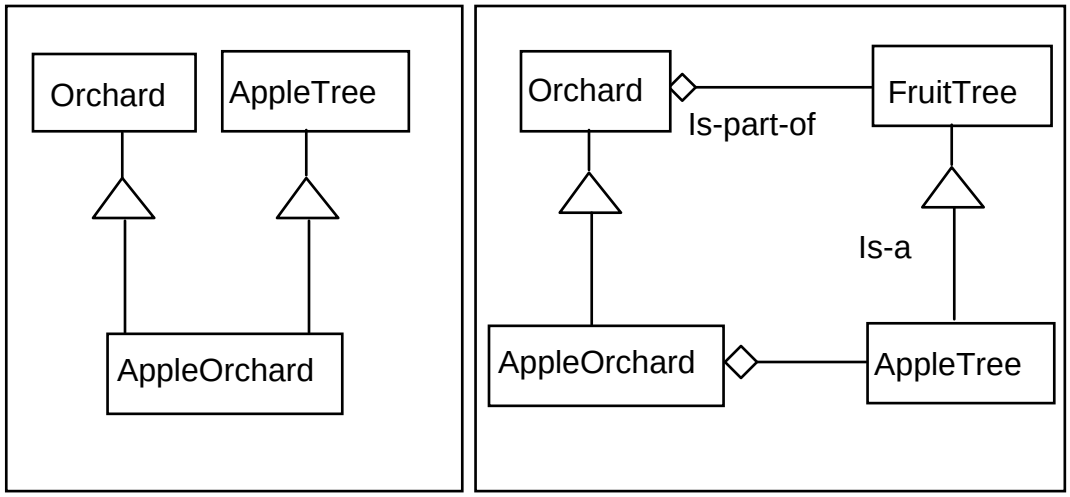


Figure 4: Example of modeling aggregation with multiple inheritance

All these approaches to composition of objects have advantages and disadvantages. None is better than the other, the respective advantages depend on the situation being modeled and on the purpose of the modeling. Certainly, the integration of the advantages of these approaches into a unique approach to composition of objects is a desirable objective. If this

[2]For more details on these problems, the reader is refered to [Cargill 91, Rumbaugh 93, Sakkinen 89].

integration is done in such a way that it reduces the disadvantages of the merged approaches then the resulting approach is promising and it may be targeted for standardization. As paradoxical as it may seem, by looking at the literature, we notice how far away we are from this goal.

1.3. Behavior aspects of composition

It is sometimes forgotten that research on composition of objects addresses two aspects, the structural aspect and the behavioral aspect. The structural aspect borrows concepts from cognitive science and emphasizes the is-part-of relationship for describing composite objects in Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Computer Aided Software Engineering (CASE), Computer Integrated Manufacturing (CIM), and Network Management (NM) areas. For the behavioral aspects, some researchers try to integrate process algebra with the object paradigm in order to compose objects like processes. To the opinion of the authors, the two aspects are interrelated. The structure of objects guides the behavior.

In the context of the IGLOO project [Bochmann et al. 92], we wish to provide a framework for the description of the behavior of composite objects. Our work is motivated by three situations for which existing approaches are not suitable to describe behavior of composite objects. These situations are the top-down design of objects, the description of frameworks, and finally the verification of certain multi-object properties. Let us illustrate the most prominent aspects of these situations which are important for the understanding of the behavior of composite objects.

1.3.1 Object-oriented top-down design

Top-down design is a well-known approach to software construction [Pressman 93]. It proceeds by stepwise refinement of designs into more detailed designs. Applied to objects, it consists on identifying the main objects of an application. Next, the inner workings of these objects are detailed. Some objects may be decomposed into component objects. Using the HOOD notation [Robinson 92], Figure 5 illustrates this process for the design of a PBX. An arrow represents a *uses* relationship, e.g., *create-connection* uses *create-trunk-connection* for its implementation. The process has three steps in this example. During the first step (a), a global object PBX providing *create-connection* and *disconnect-connection* operations is identified. Next (b), the inner working of the PBX is defined in more detail. Three component objects are defined which assist the PBX in the provision of the *create-connection* and *disconnect-connection* operations. These component objects are the trunk-

card, the conference-card and the intercom-card. In the last step (c), one component object, the trunk-card is further refined by the identification of another component object, the port to central telephone exchange.

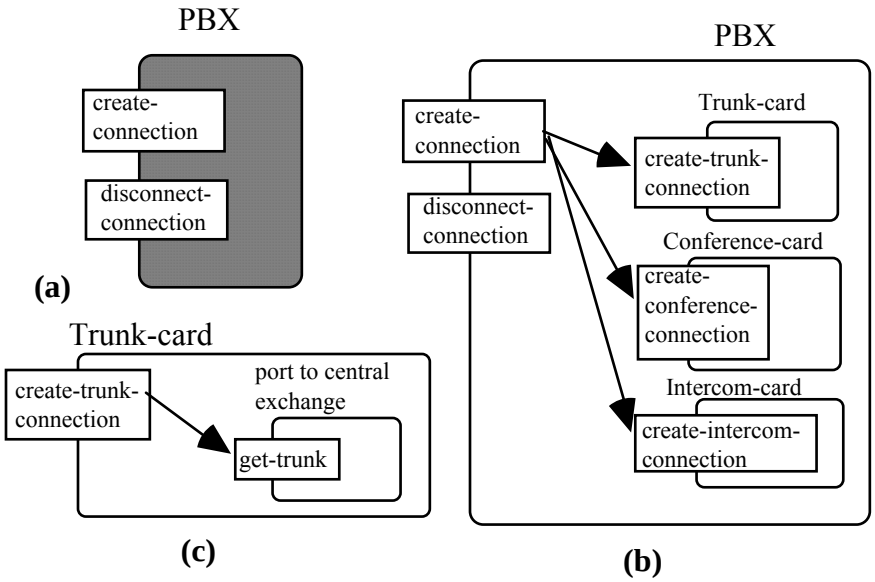


Figure 5: Top-down design example of a PBX

In many available object-oriented methods, the relationship between {create-connection, create-trunk-connection, create-conference-connection, get-trunk} is implicitly captured by object interaction diagrams [Coleman et al. 94]. Object interaction diagrams simply capture the sequencing (calling relationships) between operations. However, in the context of top-down design, the semantics of the relationship between {create-connection, create-trunk-connection, create-conference-connection, get-trunk} is richer than the semantics of a calling relationship. The way responsibilities (services) are assigned to components is of concern. The decomposition into finer objects determines the type of component objects and their relationships. Such information is not delivered by object interaction diagrams. Further, if some assertions (preconditions, post-conditions and invariants) are to be maintained by create-connection, we have to reflect these constraints on the operations {create-trunk-connection, create-conference-connection, get-trunk}. For example, a constraint may say that a trunk connection cannot be involved in more than one conference connection. This aspect is not addressed at all by object interaction diagrams.

Hopefully, the technique of contracts [Helm et al. 90] may handle these aspects. With a contract approach to design, the distribution of responsibilities to component objects is represented by **supports** clauses and assertions are also available. In the case we wish to have a concurrent participation of component objects in the provision of the operations of the global object, i.e., concurrency between create-trunk-connection and create-conference-connection, we need multi-object interactions. This is not supported by contracts. Our IGLOO project focuses on the description of distributed applications and network management applications. In such context, concurrency is a basic requirement [Bapat 94] [Forman 87].

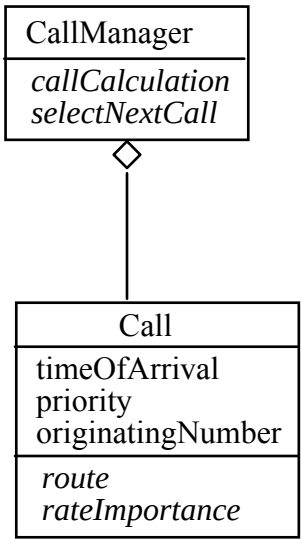
1.3.2. Framework specification problem

Frameworks are groups of classes that collaborate to fulfill certain functions. The description of a framework consists of the description of the classes and the interfaces between them. A framework records design decisions and organizes them in a set of classes related by client/server, whole/part (is-part-of), and subclass/superclass relationships. Organization of design decisions results in functional allocation inside the framework. This allocation constitutes one of the key intellectual challenge of software creation and is far more difficult to create or recreate than code.

Frameworks are identified and specified for reuse purposes. A good notation for frameworks should explicitly record and document the functional allocation inside a framework, and collaborations among classes. Most of the time, frameworks are informally described. The functional allocation and collaborations are specified using natural language annotations. It appears that objects that will be created to instantiate a framework may need to maintain some multi-object constraints, particularly, constraints on operation triggering among objects.

We conclude that the specification of a framework is analogous to that ofthe inner working of a composite object in the context of object-oriented top-down design as described before. To illustrate this, let us take the example of a *callManager* object handling several *call* objects (see Figure 6, taken from [Coad 92]). In this example, the framework consists of two classes related by an is-part-of relationship. The design decision of making the component objects do as much as they can with what they know is implicitly recorded in this framework. *Call* objects are allowed to route themselves and to determine the importance of the call using some call-specific information. To select the next call to process, we need to compare the rate of the calls which are pending. To this end, the

information about the rate of the pending calls should be gathered. To enforce separation of concerns, the *callManager* object is allowed to determine which call gets to go next. By achieving separation of concerns, the role fulfilled by each participating object (*call* or *callManager* objects) is well-defined and extensible.



A "call" object knows its time of arrival, priority, and its originating number. The "call" object also knows how to route itself. It can even rate its importance.

But the "call" object does not know enough to make the actual selection which determines which call gets to go next. The callmanager object is used for this purpose, based on a selection algorithm, it is responsible for selecting the next call.

Figure 6: A callmanager object handling several "call" objects

For reuse purposes, this design decision of localizing the behavior should be explicitly described in the framework specification. In addition, the *callCalculation* and *selectNextCall* operations should be documented. Some *call* objects are related to the same *callManager* object which use these objects to select the next call to be processed. Once a call is selected it cannot be selected again. How this is reflected in the framework specification is a matter of concern.

1.3.3. Verification of certain multi-object properties

The verification of certain multi-object properties arises when we wish to reason about the composite object behavior based on the behavior of component objects [Abadi, Lamport 93]. This is motivated by problems encountered when dealing with incremental composition (stepwise formation of composites) in application areas such as CAD, CAM, CASE, CIM, and NM. Here the behavior of the resulting composite object should be predicted according to the behavior of the component object being added, modified or

removed. For example, using the PBX example, if the card providing the trunk connection facility is removed, what happens to the operations provided by the PBX[3]?

In this case, we can imagine that the composition mechanism used for forming the PBX may state that when a component is removed, operations it provided remain undefined. However, such handling of component removal in an application requires support of composition mechanisms and associated rules by the programming language. This may not be achieved without identification of such mechanisms and associated rules. We believe that such composition mechanisms and rules may be derived by analyzing generic patterns of composite objects. These mechanisms and rules should be recorded along with each composite object to characterize its composition. Next, we need to understand how these mechanisms and rules work and what is required to automatically support such mechanisms and rules. Another aspect of multi-object properties is that relationships among objects have to be made explicit in order to verify the consistency of multi-object properties associated to these relationships. In addition, formal specification languages are used to describe the behavior of objects. We are interested to examine how formal specification languages contribute to the verification of multi-objects properties

1.4. Purpose of this paper

We have shown that existing approaches to composition of objects are defined for specific contexts. The diversity of the available approaches originates from multiple interpretations of the concept of object composition. In fact, the definition given in the literature of composition of objects is amenable to the four interpretations described in Section 1.2. This creates confusion for the use of this term. We believe that the major issue in composition is *providing a non-ambiguous, and workable definition of object composition and the composition process which covers the aspects of object composition*. Among these aspects, we address in this paper particularly those aspects related to the **behavior** of composite objects, such as:

• How are the responsibilities assigned to component objects?
• Coverage of interactions among component objects
• Explicit description of the semantics of these interactions

[3]Few will dispute the claim that this problem is related to feature interactions in telecommunications systems which is more complex than it appears, see [Cameron, Velthuijsen 93] for instance.

• What are the design decisions which lead to the assignment of some responsibilities to components ?
• What are the rules which govern insertion and removal of a component object in a composite object?
• Are there some canonical forms of composition?
• If such rules and canonical forms exist, then each composition should mention what are the rules it follows and what is its semantics in terms of canonical forms of composition.

Composition of objects is an invaluable tool for structuring applications through abstraction of chunks of objects and by capturing the decomposition structure of objects. It also captures an important aspect of the world, i.e., many things of this world are composed of other things. In this respect, the achievement of a unique approach to composition of objects, like the standardization of the meaning of subtyping (inheritance), is a desirable objective. This paper suggests such an approach. Contrary to available approaches, it handles both structure and behavior of composite objects.

We propose an approach to composition of objects which uses an ontological definition of a composite object. A composite object is formed by superposing three kinds of properties: inherent, aggregate and emergent properties. Inherent properties are the properties of the component objects which are visible without any change at the composite object level. Aggregate properties are the properties created by the composition process; an aggregate property represents the aggregation of analogous properties of component objects. Here aggregation is taken in the sense of summation, integration, functional composition or other means for aggregating properties of objects. Emergent properties are the properties which although created through composition, cannot be predicted from the properties of component objects. It should be noted that composition does not alter the component objects.

We also describe a composition process. The basic idea is to achieve separation of concerns through three steps, each focusing on a particular aspect of the composite object. The first step relates to how component objects are configured to form the composite object. In this configuration process, the relationships among objects and their associated constraints are of concern. The next step, called juxtaposition, relates to the composite object and its relationship with its component objects. During this step, inherent and aggregate properties are identified and described. Inherent and aggregate properties are related to both composite and component objects. Finally, the step of emergence deals with emergent

properties and other refinements to the composite object. While new in this specific form, this approach is related to existing approaches to object composition. Further, it acts as an integrator of various existing approaches to object composition due to its ontological grounding and abstract approach to composition.

Many aspects pertaining to the structure of composite objects have been reported elsewhere in the literature; we recall these aspects and introduce new ones when necessary. New aspects pertainning to the behavior of composite objects are presented. These aspects are supported by fundamental principles defining the behavior and interaction among composite objects. The remainder of this paper is structured as follows:

In Section 2, we define the concepts of composition and composite object. We begin by precising the notion of object and associated properties. Then, we motivate the necessity of resorting to Ontology in order to define composition and composite objects. Composition and composite objects are defined according to an ontological perspective. Using the same perspective, the notion of is-part-of relationship is introduced. It is compared to the notion of is-part-of relationship based on linguistic, logic and cognitive sciences. From this comparison, an improved definition of the is-part-of relationship results. Next, some aspects of composite objects are illustrated through an example. We close the Section 2 by an overall discussion motivating the distinction between different kinds of properties for composite objects, namely, resultant and emergent properties. Resultant properties are further subdivided into inherent and aggregate properties.

Armed with these concepts, we introduce in Section 3 a framework for object composition. The framework consists of the concepts presented in Section 2 and a composition process. The composition process is presented by outlining three steps. These steps are configuration, juxtaposition and emergence. We then illustrate through an example the use of the framework for describing composite objects. Finally, a short discussion on the integration of this framework in existing OOADM closes Section 3. We conclude this paper by recalling the theme explored, highlighting the contributions of this work, and pointing to future developments.

2. Towards a definition of composition

2.1 Notion of Object

Before working towards the definition of composition, let us introduce what we mean by an object. We view an object as an abstract or concrete entity (or thing) characterized by its (observable) properties. Intuitively, a property is an (observable) aspect of an object. By a property of an object, we mean features such as:

- having an attribute
- having an attribute value
- having an operation
- displaying a behavior
- being subject to constraints, etc

Introducing the notion of property allows us to handle both structure and behavior of objects in an abstract manner.

2.2 Necessity of resorting to Ontology

We have mentioned in the introduction that there exist several interpretations of the concept of composition of objects. In fact, the word composition is a *buzzword*; it is used with many meanings. From the brief survey of the literature given before, we note that, depending on the aims of specific researchers, a given understanding of composition emphasizes certain aspects and neglects others.

As already mentioned by Wand [Wand 88] , certain overused object-oriented concepts, like inheritance, object, attribute, relation, ... have a common sense meaning. A specific understanding of such concepts may be constraining. To illustrate this constrainment, let us consider the following aspect of properties: *"Properties belong to objects and properties may not exist without an object to which they are attached to"* [Bunge 77]. This characteristic of properties prohibits modeling of properties of objects as objects. Adopting such a principle forces the object model to have a specific construct different from the object construct for modeling properties.

In order to properly define such concepts, Wand proposes to refer to ontology of science to clarify the terminology used, and to derive the implications of a particular understanding of

those concepts. Why should we refer to ontology? Ontology is the branch of philosophy of science dealing with modeling the existence of things, it plays an important role in the axiomatic foundation of scientific theories (see for instance [Bunge 77, Bunge 79]). If a paradigm like the object paradigm is axiomatized (i.e., definition of basic concepts and relations among them), some of the following concepts are used explicitly: part, juxtaposition, property, composition, state function, state, event, relationship, process, space, ... However, the specific axioms of the object paradigm will usually not tell us anything about such fundamental and generic concepts; the paradigm just borrows them leaving them in a intuitive or presystematic state. Only ontology is interested in explaining and systematizing concepts which, while they are used by many sciences, are claimed by none. Ontology can render the service of analyzing fashionable, but obscure, notions like the composition of objects.

Another motivation for using ontology is that the composition process is complex; one has to deal simultaneously with several aspects related to a given composition, and the interactions of these different aspects are not fully understood. If an approach chosen for composition is based on fundamental principles, it may harmoniously deal with the interrelated aspects. Ontology offers basic principles which are useful for dealing with complexity. Further, ontological principles are useful to object-oriented analysis and design with composition, due to a common objective. Both try to make a model (picture) of the real-world.

2.3. Composition from an ontological perspective

Composition is a mechanism for forming an object from others objects. The object resulting from the composition is called a composite object (composite for short) and the objects which were composed are called component objects (components for short). In other words, An object x is composite if and only if there exist objects y and z such that $y \neq x$, $z \neq x$, and y is a component of x and z is also a component of x. Otherwise x is a simple object.When an object y is a component of a composite object x, the relationship between y and x is called a *is-part-of (part/whole)* relationship.

Ontology proposes many principles related to composition of objects. The first principle defines what we mean by structure of a composite object.

Principle 1: A composite object has a structure, the organization of its component objects and the relationships they have with the composite object.

The structure differentiates the composite object from a simple object. A composite object without structure is a simple object. This is guaranted by the definition of a composite thing given by Bunge [Bunge 77].

> *A substantial individual (thing) is composite (or complex) iff it is composed additively of individuals (things) other than itself. Otherwise it is simple (or atomic or basic).* [4]

This structure is further refined into is-part-of and inter-object relationships among the component objects.

2.4. The is-part-of relationship

The is-part-of relationship is characterized in Ontology by the following principle:

Principle 2: The is-part-of relationship is a partial order relation.

This means that the is-part-of relationship is *reflexive, asymmetric, and transitive.* The is-part-of relationship has been thoroughly examined in the database community. In the following, we summarize the major findings:

(a) Is-part-of relationships are divided along the lines of *exclusive* and *shared*. An exclusive is-part-of relationship enforces the restriction that a given object can be a component of only a single composite object. A shared is-part-of has no restriction on the containment of an object in several composite objects.

(b) Is-part-of relationships may have a cardinality of one or many. A cardinality restricted to one means that the composite object is allowed to have only one component object of this type. A cardinality of many allows the composite object to have several component objects of the same type.

[4]In this definition, reference to additive composition comes from the debate on what is important in the description of a composite? The structure of the composite or its properties. When the structure of the composite is of prime importance, its description tends to emphasize its components and their properties. The composition is viewed as the addition of components resulting in a misleading terminology of *additive* composition. Otherwise, the properties of the composite are described without any relation to their origin, *multiplicative* composition.

(c) In a composite object, is-part-of relationships may be ordered, e.g., documents consisting of ordered paragraphs and sections.

(d) Is-part-of relationships are also divided along the lines of dependent and independent. A dependent is-part-of relationship means that the component object should be deleted when the composite object is deleted. Otherwise, it is an independent is-part-of relationship.

(e) An is-part-of relationship is said to be value or function propagating if some constraints should be propagated from the component objects to the composite object or the reverse.

2.5 Another view of composition

In a recent paper, Odell [Odell 94] proposes a taxonomy of composition relationships. This taxonomy is borrowed from a study in linguistic, logic and cognitive psychology by Morton Winston, Roger Chaffin and Douglas Herrman [Winston et al. 87]. The composition relationships may be classified depending on whether they belong to the following classes:

Configurational relationship: the component objects bear a particular functional or structural relationship to one another or to the composite object.

Homeomeric relationship: the component objects are of the same type as the composite object.

Invariant relationship: the component objects can be separated from the composite object.

The following six kinds of composition can be distinguished:

	Configurational	Homeomeric	Invariant
Component-integral object	Yes	No	No
Material-object	Yes	No	Yes
Portion-object	Yes	No	No
Place-area	Yes	Yes	Yes
Member-bunch	No	No	No
Member-partnership	No	No	Yes

In addition, we may introduce the distinction between *extensive* and *non extensive* composite objects. A composite object is extensive when the component objects are physically included in the spatial volume occupied by the composite object they form; otherwise it is a non extensive composite object.

As one may notice, there are different aspects related to composition and is-part-of relationships: exclusive/shared, cardinality (one, many, fixed), dependent/independent (invariance), value propagating, configuration (ordered), homeomerous, and extensive. Note that this list is not exhaustive.

2.6 Exemplification of some aspects of composition

Before going further in detailing composition, let us illustrate some of its aspects through the example of a part of a system for tracking container movements through docks: the containment of a car in a container object. This is shown pictorially in Figure 7, along with an object diagram of the application.

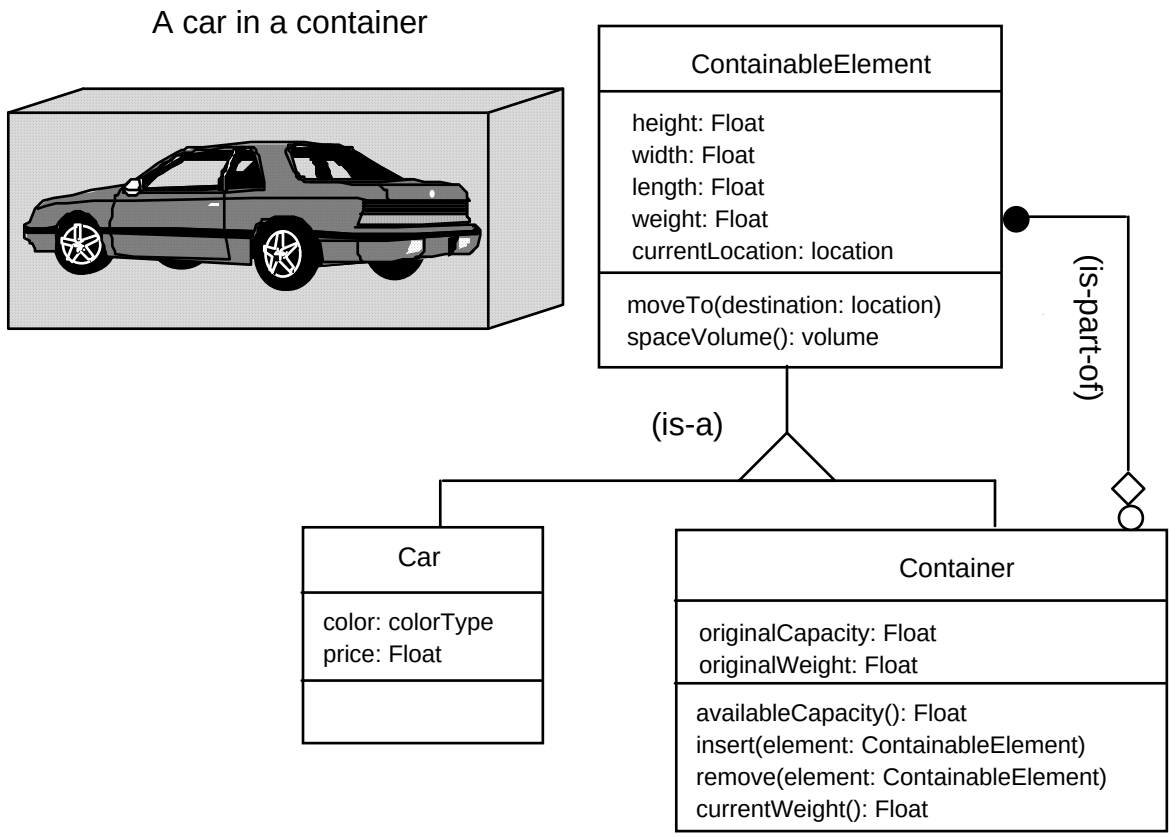


Figure 7: Object diagram of car and container

When the car is stored into the container, it becomes a component of the container, i.e., an is-part-of relationship is established between the car and the container. According to [Odell 94, Winston et al. 87, Halper et al. 92], this is-part-of link is:

(i) *component/integral object* due to the particular structural arrangement in relation with other possible components of the container. The components should fit into the container, i.e., the sum of the component volumes should be less or equal to the capacity of the container. This has the following implications:

- ∅ The total weight of the container depends on the weight of its components.

  $\text{ContainerWeight} = \text{Original Weight} + \sum_{i}^{n} \text{Component Weight}$

  where original weight is the weight of a container without components.

- ∅ The available capacity of the container depends on the volume of its components.

$$\text{AvailableCapacity} = \text{Original Capacity} - \sum_{i}^{n} \text{Component Capacity}$$

where original capacity is the capacity of a container without components.

(ii) *exclusive/independent:* the car should not be shared by containers and the destruction of the container object does not necessarily leads to the destruction of the car object;

(iii) *extensive* in the sense that the container occupies a volume of space and its components are (physically) included in this spatial volume. Thus, insertion and removal of a component object should obey to the rules:

- ∅ The insertion of a component causes the moving of this component inside the spatial volume of the container object, and the container weight and capacity should reflect this situation.

- ∅ The removal of a component causes the moving of this component outside the spatial volume of the container object, and the container weight and capacity should reflect this situation.

(iv) *function propagating:* certain operations of the composite object are also uniformely applied to component objects, e.g., moving the container object.

- ∅ The moving of the container object to a new location causes the moving of its components, too.

As one may notice, with this simple example of a container object, there are many aspects to be taken into account when dealing with composition. In the following, we go further in the characterization of composite objects by explaining their properties.

2.7. On resultant and emergent properties

According to Bunge [Bunge 77], ontology establishes that a composite object has emergent and resultant properties. This is explicitly stated by the following principle:

Principle 3: A composite has resultant and emergent properties.

Resultant properties are inherited (or derivable) from the component properties. Emergent properties are higher order properties resulting from the bundling of the components. The latter properties are not derivable from the components properties. Certain researchers concentrate on the resultant properties due to their predictability, and neglect the emergent properties. Others prefer emergent properties because they are novel.

The debate on emergent versus resultant properties has divided the researchers into *reductionists* and *holistics*. Reductionism, is an epistemological doctrine according to which the study of a system is reducible to the study of its components. Holism is the ontological view that stresses the integrity of systems at the expense of their components and the mutual actions among them. The reductionists state that the composite is completely defined by its components while the holistics claim that the composite is more than the sum of its parts.

Adopting one or the other view has some impact on the way we should handle the composition process. If we adopt a reductionist standpoint, then all the properties of the composite are derivable from the properties of the components. The composition becomes deterministic and predictable. This standpoint is useful if we limit ourselves to composites which have no significant emergent properties. Composition becomes a sort of logic where some operators (composition "rules") allow the composition of objects. These operators are deduced from generic "patterns" of composition. This view is supported by ODP [ODP 93], Lam & Shankar [Lam, Shankar 92], Zave & Jackson [Zave, Jackson 93], and Abadi & Lamport [Abadi, Lamport 93].

In certain application domains, we have to focus on the new properties acquired by a system (emergent properties). For example, in the management of distributed systems, the system is structured according to the scheme "manager-agent" [Yemeni 93]. A manager controls several agents and an agent monitors a given domain of the system. Combining the domains monitored by agents to get domains under the control of a given manager, results

in a bigger domain with properties which are often more than the sum of the properties of its component domains. In such conditions, where we have to handle both resultant and emergent properties, a view combining both reductionist and holistic approaches is necessary. Fortunately, by achieving separation of concerns we are able to mix both approaches. The resultant properties are handled by a reductionist approach and the emergent properties by an holistic approach.

Resultant properties can be further classified into *inherent* properties and *aggregate* properties. Inherent properties are the properties inherited as such from component objects by the composite object, i.e., without any change in their semantics. They are of two kinds, basic and composite. A basic inherent property can be attributed to only one component object while a composite inherent property may be the combination of properties of several component objects. This combination is named at the composite object level and regroups many basic inherent properties. On the other hand, aggregate properties are created by the composition process; an aggregate property represents the aggregation of analogous component object properties. Here aggregation is taken in the sense of summation, integration, functional composition or other means for aggregating properties of objects. A relation can be established between aggregate properties and some inherent properties. Most of the time, aggregate properties can be expressed in terms of a mathematical relation to inherent properties. Using the container example, the length of the container is an inherent property and its current weight is an aggregate property.

The two categories of resultant properties and the emergent properties are different in nature. Inherent properties are properties of component objects, and exist before the composition process. We claim that emergent properties are independent of resultant properties and this assumption is supported by the following ontological principle, borrowed from the Theory of emergence [Ablowitz 39] and [Angyal 39], which links emergent properties to resultant properties but clearly establishes the distinction between them.

Principle 4: Emergent properties are rooted in the resultant properties but are not reducible to them.

To summarize, we have found that the presence of component objects differentiates simple objects from composite objects. These components are organized in a structure which

specifies the interactions among the components. Superimposed upon this structure is the is-part-of relationship which relates component objects to the composite.

Due to the presence of component objects, the composite offers certain properties of these components. These component properties are called inherent properties of the composite. The composite object may also organize the properties of components into more complex properties called aggregate properties. In addition to the above properties, novel properties (emergent properties) may be defined for a composite. Emergent properties are not attributable to the components. What really makes a composite different from a simple object is the possible presence of inherent and aggregate properties. As a consequence, a composite object showing only emergent properties can be treated as a simple object. Furthermore, we distinguish between the inherent and aggregate properties. This distinction is grounded on the existence of inherent properties prior to the composition process, while aggregate properties depend upon the composition process. This leads to the idea of superposing these properties in order to form composite objects. As a consequence, *we model a composite object as an object characterized by three kinds of properties: inherent, aggregate and emergent properties.*

This research concerns the description of composite object behavior. To this end, we shall precise in the sequel the notion of behavior for composite objects. A behavior is a property of an object. As such, it can also be partitioned into three categories of behaviors.

The behavior of an object is the observable (re)action (response) it provides (to a given stimulus). This is a very abstract view of behavior. In other words, we may define a behavior as a collection of actions with a set of constraints on when they may occur. Note that a behavior may include internal actions. Action triggering at an object is done through message passing. Each action produces an effect. It may: (i) change the current state of the object which performs this action, (ii) change the environment of the object (action initiator or performer), or (iii) return an object (or value) or set of objects (or values) to the object which initiates this action. Using this meaning of behavior and according to the three kinds of properties for a composite, we define three kinds of behaviors:

(1) **Inherent behavior:** An inherent behavior of a composite object is the collection of actions with associated constraints which are defined by some components of this composite. The inherent behavior is such that the actions and constraints defined at component level are the unique actions and constraints which define the semantics of this

behavior. In other words, it is a behavior defined at the component level which is mediated by the composite object without any modification to its semantics. The syntax of this behavior may be changed to cope with name conflicts if there is more than one inherent behavior of the same kind offered by the composite. An inherent behavior can be considered as the basic behavior that the composite may offer when it makes some of its components visible to other objects.

(2) **Aggregate behavior:** An aggregate behavior of a composite object is the collection of actions with associated constraints such that this behavior is formed using aggregation (composition) of behaviors of components. A behavioral aggregation (composition) is a combination of two or more behaviors yielding a new behavior. The characteristics of the resulting behavior are determined by the behaviors being combined and the mechanism of behavioral aggregation used. Examples of behavioral aggregation mechanisms are sequential aggregation, and concurrent aggregation. In this latter case, the composite does not mediate the behavior of some of its components. The aggregate behavior is defined at the composite level. It makes use of low-level behaviors. The composite coordinates the components for the provision of the aggregate behavior.

(3) **Emergent behavior:** An emergent behavior of a composite object is the collection of actions with associated constraints which may refine existing actions and associated constraints or define totally new actions and associated constraints. In other words, the emergent behavior may refine existing behaviors (inherent or aggregate) or define a totally new behavior.

3. A Framework for Object Composition

After providing a model for composite objects, we propose a model for the composition process. A composition process helps to construct composite objects. Our starting point is the basic idea that a composite object is formed by superposing three kinds of properties. Intuitively, superposition of properties means to put together a set of properties in such a way that the interrelations between these properties are kept minimal and the whole remains coherent. If the composition process has to support this principle, then a stepwise process to composition is recommended. This reasoning is also backed by observing that some composite object descriptions may be too complex and we need to focus on certain aspects of composition. We therefore implement separation of concerns in the composition process, allowing different aspects of a composite object to be treated separately. This has the advantage of localizing related aspects.

Some aspects of a composite object are built upon others. This is the case for inherent and aggregate properties. The general principle of cohesion dictates that closely related ideas should be kept together and unrelated ideas be kept separated. Thus inherent and aggregate properties should not be treated independently. Also, following the idea of cohesion, the interactions among the component objects and the overall properties of the composite object should be treated distinctly. A composite object may have properties which are not attributable to component objects.

Taking into account the aspects of composite objects presented above, there appears to be three distinct aspects of a composite object: (1) its internal activity, (2) its relationship with its component objects and resulting properties, and (3) the properties non attributable to its component objects. For the definition of each aspect, we assign a specific step. The proposed process consists of the three consecutive steps.

3.1. The composition process

We will now give an overview of each step and explain how they are interrelated. The first step has to handle the interactions among the component objects. In other words, it has to deal with the internal activity of a composite object. We call this step *configuration of component objects* (or configuration for short). The second step, which we call *juxtaposition,* has to describe the resultant properties. It has to deal with the way the composite object is handled as a unit, i.e., it concerns the identity of the composite object,

the extension of this identity to its component objects and their encapsulation in it. All the aspects of this step are tributary to the is-part-of relationship. Finally, the *emergence* step does not depend on the is-part-of relationship. It concerns only properties which cannot be attributed to component objects. Note that this step corresponds to describing an object by specialization.

Configuration

Component objects may bear relationships among one another. These relationships constrain the way the component objects may associate with one another to form the composite object. This constrainment is called the configuration of the component object. It is characterized by the relationships mandatory for the composition of the component objects and their related constraints.

These relationships also characterize the component objects. It follows that the properties of component objects and the mechanisms for configuring them cannot be dissociated. Following this idea, the relationships among the component objects should be made explicit as a characteristic of the composite object. For this purpose, we define the context of an object as the set of relationships it has with other objects. The set of relationships may have associated constraints which prescribe the properties of this object. In other words, configuring objects means to associate through relationships "plug-compatible" properties of these objects. This leads to considering the process of selecting how the component objects may be related. This selection process may be dictated by the designer or by the domain being modeled. As such, the way component objects are configured captures phenomena of interest according to the domain being modeled or design decisions characterizing an implementation artifact chosen by the designer.

The constraints associated to the relationships among the component objects define (possible) interactions among component objects. To cope with these multiple aspects of a configuration, we introduce the notion of *role* played by an object in a configuration. A role is the mandatory set of relationships and associated constraints an object must have to participate in a configuration. This concept of role captures some phenomena of interest and design decisions as explained before. A role can be played in multiple contexts. Objects may have multiple roles in the same context. A role captures the define/use patterns for each component object.

Finally, there may be interactions among roles in a configuration. These interactions may be mediated by relationships or their associated constraints. Such related roles form a structure of interacting roles. Thus special attention should be given to interactions among roles and the consistency of a set of interacting roles.

To summarize, the configuration of component objects consists of the definition of the role of each component object and explicit specification of the semantics of the interactions among these roles. In addition, constraints may be imposed on all the roles. A role is described by the set of define/use properties associated with the role. The interactions among the roles are abstracted through relationships.

Juxtaposition

Juxtaposition uses the outcome of the configuration step to build a composite object. The basic idea underlying this step originates from the following fact claimed by Tversky and Hemenway [Tversky and Hemenway 84]:

> *Names of parts frequently enjoy a duality not apparent in other attributes; they refer both to a perceptual entity and to a functional role.*

We claim that the functional role[5] of a part (component object) captures certain phenomena of interest. These phenomena are reflected in both inherent and aggregate properties defined for the composite object. Inherent properties result from the conveyance of component objects properties to the composite object level. Before defining aggregate properties, the characteristics of the is-part-of relationships superimposed on the configuration, their implications for the component objects, and the visibility of component objects at composite object level are to be specified. Aggregate properties are the properties created by the composition process; an aggregate property represents the aggregation of analogous component object properties. Here aggregation is taken in the sense of summation, integration, functional composition or other means for aggregating properties of objects.

[5] Civello [Civello 93] introduces the notion of functional composition and non-functional composition. A composition is functional when the component has a specific role to fulfill which allows it to participate in certain operations of the composite. In non-functional composition, the component is like an element of a set (the whole) in the mathematical sense of the term.

To summarize, the juxtaposition step consists of three phases: (i) identification of the characteristics of the is-part-of relationships binding each component to the composite; (ii) determination of the visibility of components and their properties, thus determining the inherent properties of the composite object; (iii) definition of the aggregate properties. These phases determine the functional role played by each component in the composite.

Emergence

The emergence step consists of the extension of existing properties or the definition of new ones. This is a case of specialization. These new properties must be compatible with existing properties. Property extension follows subtyping rules, and consistency checking in relation to existing properties is done according to these rules.

3.2. An example of composition

In this subsection, we illustrate the composition with a simplified version of the lift problem [IWSSD-4]. We omit certain details which do not contribute to the essential points of this paper. The statement of the lift problem is:

A lift system is to be installed in a building with m floors. The lift is only aimed at moving goods from one floor to another. Persons are not allowed to ride in this lift. Therefore, there are no control buttons inside a cab. The cab is allowed to support a given maximal load depending on the value of the counter weight and the power of the motor. A sensor determines if the allowed maximum load is not violated. If so, the sensor sends an overload signal (overload warning) to the lift control mechanism. It is assumed that the lift and the control mechanism are supplied by the manufacturer. The internal workings of the control mechanism are not of concern. The aspects to be described concern the usage of this lift by a clerk. The lift is used under the following constraints:

1. Each floor has buttons: floor selection buttons, door command buttons, and lift request buttons. These buttons illuminate when pressed. The illumination is canceled using the following rules:
- for floor selection buttons, when the lift reaches the desired floor
- for door command buttons, when the operation is completed
- for lift request buttons, when a lift visits the floor and is either moving in the desired direction, or has no outstanding requests. In the latter case, if both floor request buttons are pressed, only one should be canceled. The algorithm to decide which to service first should minimize the waiting time for both requests.

2. All requests for lifts from floors must be serviced eventually, with all floors given equal priority.

Assumptions made for this problem are:
- the cab has a fixed original volume which limits the quantity of goods which may be transported using the lift.
- no clerk is allowed to ride in the lift.

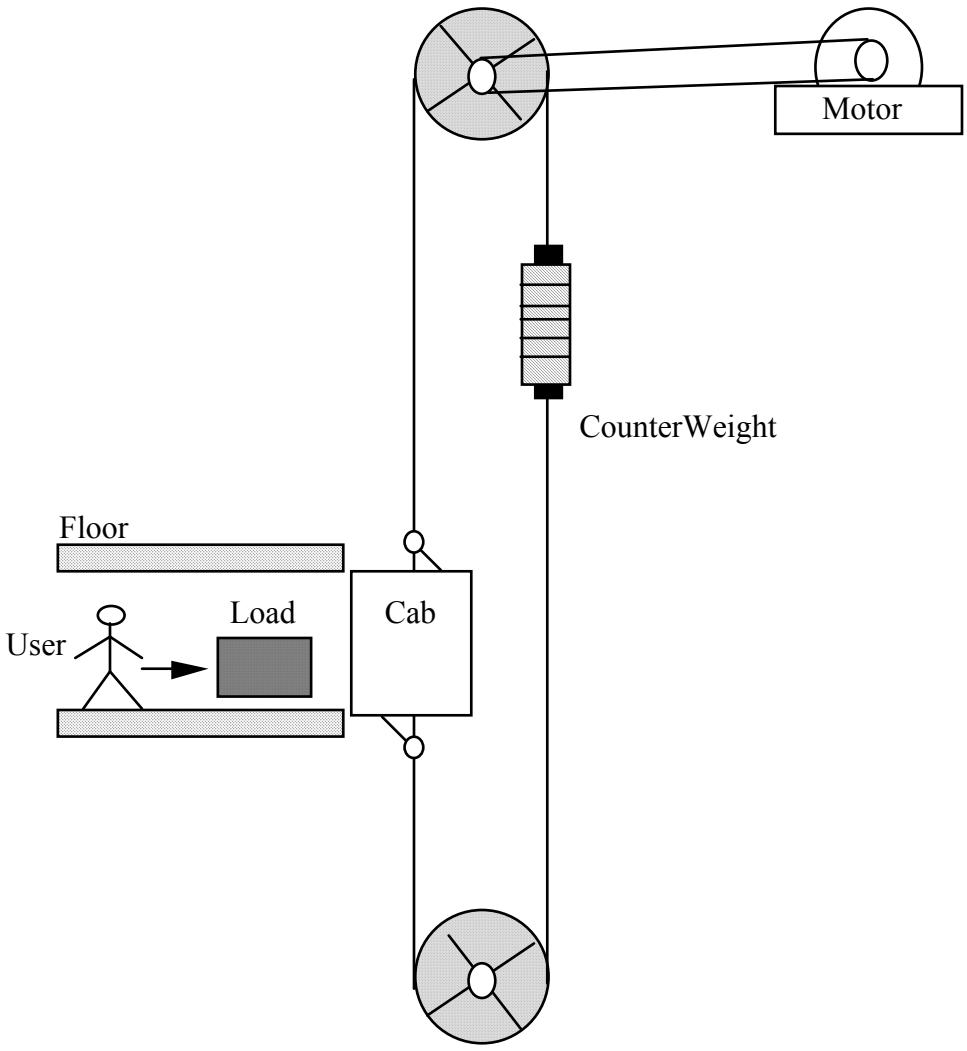


Figure 8: Lift problem

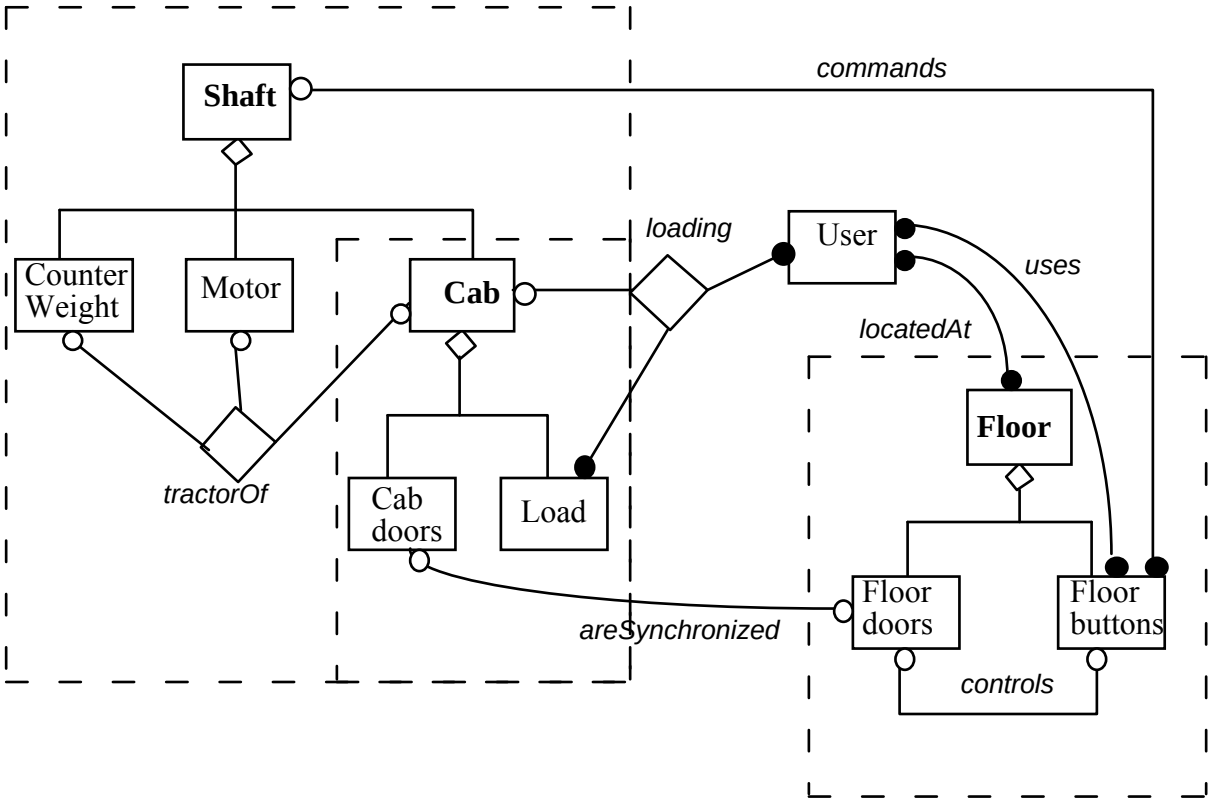


Figure 9: Object model of the lift problem

The problem domain is represented in Figure 9 using the OMT object type diagram. In this diagram, the classes are depicted as boxes labeled by the class name. The lines with a diamond at the end represent is-part-of relationships. The other lines between classes represent associations. The circles on associations denote cardinalities; a plain circle denotes a cardinality of one-to-many; an empty circle denotes a cardinality of zero or one. Ternary associations are depicted by rhombuses.

We can identify three composite objects (indicated in Figure 9 by dashed rectangles): (1) the cab possibly with goods inside, (2) the shaft, and (3) the floor. Note that the composition of these three composite objects, in turn, form a larger composite, namely the lift system. Therefore, the lift system can also be modeled by a composite object. It results in the diagram of Figure 10. In the sequel, we follow the proposed composition process to describe the cab, floor and shaft composites.

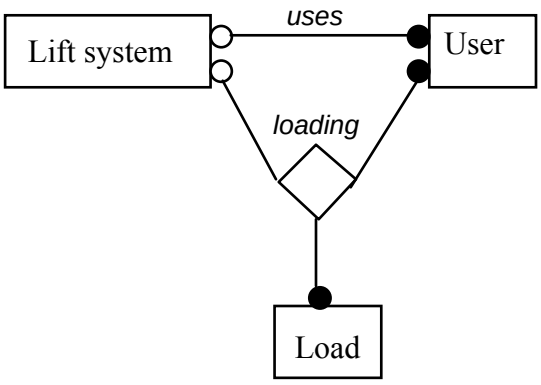


Figure 10: The lift system composite object

Cab composition

A cab is made of a frame and doors. It may carry goods. From a user perspective, only doors and goods are of concern. However, the frame with its fixed size restricts the capacity of the cab. We also assume that cab doors are synchronized with the floor doors when the cab arrives at a given floor. With these considerations in mind, let us now go through the composition process.

(1) **Configuration** (analysis of interactions among the component objects): There is no interaction of interest among goods carried by the cab and the cab doors.

(2) **Juxtaposition** (individuation of the cab as a composite object and determination of the inherent and aggregate properties): The individuation of the cab object concerns the mechanisms used to create a cab in a given application. At this level of description, this is of no interest.

The inherent properties are those component properties that are available at the composite object level. *As a rule of thumb, one may say that the properties of visible component objects become automatically the inherent properties of the composite object.* Visibility of component objects depends on the application at hand. In the case of the cab object, the doors are visible. It follows that door properties are visible at cab level, therefore are inherent properties. In particular, the doors provide the operations for opening and closing the doors.

Now, let us consider aggregate properties. We define aggregate properties as properties representing certain aggregations of component properties. The mechanism of aggregation

should have a well-defined meaning and provide a valuable property at the composite object level. In the context of the cab, its weight, spatial volume and available volume are meaningful aggregations at the component object level. The weight of the cab consists of the sum of weights of its components. The aggregation mechanism used here is the arithmetic sum. On the other hand, the spatial volume of the cab is a more complex aggregation of the spatial volumes. The spatial volume of a composite is the sum of the spatial volume of each component. Therefore, the spatial volume of the cab is also an aggregation of the component's spatial volume. The aggregation mechanisms used here is also summation. Finally, the available volume within the cab for loading is also an aggregate property. It is computed by substracting from the original volume of the cab the volume occupied by the loading of the cab. Here again, it can be expressed using the arithmetic calculation.

Another aspect of this step is the identification of the characteristics of the is-part-of relationships. By using the terminology introduced in Section 2, we find that all the is-part-of relationships of this composition are (a) exclusive, i.e., components cannot be shared, (b) independent, i.e., the component may exist as a separate entity outside the composite object, (c) extensive, i.e., components are included in the spatial volume of the composite object.

(3) **Emergence** (determination of the emergent properties): In the context of the cab description, there is no emergent property of interest.

Floor composition

A floor is made of doors and buttons. The clerk is located at a given floor when loading or unloading the cab. We assume that each floor is numbered. With these considerations in mind, we go through the composition process.

(1) **Configuration**: There exist interactions among doors and buttons. Doors are controlled through floor buttons. When the cab arrives at a given floor, if one button (up or down) is pushed then it may cause the doors to open. These interactions are captured through the semantics of the relationship relating doors to buttons. The precise description of this relationship implies the specification of the dynamic behavior of the components and their relationships using some suitable formalism, such as for instance Contracts [Helm et al. 90]. This is outside the scope of this paper.

(2) **Juxtaposition**: Inherent and aggregate properties are identified like in the previous composition. Inherent properties are operations related to visible component of the floor: the doors and the buttons. There is no aggregate property of interest in this composition. The is-part-of relationships are exclusive, independent and extensive for the same reasons as in the previous composition.

(3) **Emergence**: We note that the floor number is independent of the component doors and buttons; therefore, it is an emergent property. The floor number is aimed at uniquely identifying a floor. For instance, for the description of the behavior of the lift system, which represents a higher level of composition, it is important to distinguish floors in order to specify, for instance, that the cab will move to floor 5 when button of floor 5 is pushed.

Shaft composition

This composition is left as an exercise to the reader. We note that:
(a) The shaft components are the counter weight, the motor and the cab.
(b) There are interactions among these components.
(c) Cab doors are the only visible components from the user point of view.
(d) The shaft has a weight.
(e) The shaft may be put in a dormant state if an overload signal is sent.
(f) The shaft has a mean service time depending on multiple factors like the number of floor buttons depressed, the loading of the cab, the motor conditions, etc.

For each of these composite objects, we provide in the Annex an informal description capturing the ideas presented above for composition. These descriptions are made using the template depicted in Figure 11. Studies are underway in order to define a notation with precise syntax and semantics that will capture the most important aspects of composition.

<object> composition
configuration
roles:
interactions among roles:
for each role
relationship to other roles:
phenomena of interest captured by this relationship:
associated constraints:
additional constraints over roles:
juxtaposition
for each component object
is-part-of characteristics:
functional role of the component:
inherent properties:
for each inherent property
property description:
phenomena of interest captured by this property:
aggregate properties:
for each aggregate property
property description:
phenomena of interest captured by this property:
emergence
emergent properties:
for each emergent property
property description:
phenomena of interest captured by this property:
end <object> composition

Figure 11: Template for composite object description

3.3 Integration within existing OOADM

In this subsection, we highlight some aspects which must be considered in order to integrate the proposed composition framework within an existing OOADM. This kind of integration is not easy and will not be examined in detail. However, we provide some hints on how one may proceed to integrate this framework in an existing OOADM.

An OOADM is characterized by three aspects: (a) the models it proposes for understanding and designing an application; (b) the concepts which are used to describe and model the application; and (c) the development process leading to the construction of these models. Therefore, we need to examine how the concept of composite object influences these three aspects of an OOADM.

An application consists of a set of interacting objects. In order to describe a given application, the concepts underlying an OOADM should at least precise the following elements:

- What is an object?
- How objects are described?
- What are the interactions among objects and how these interactions are described?
- How objects are classified?
- How are defined the concepts of encapsulation and subtyping?

In order to integrate the proposed framework, we need to extend these elements to cope with composite objects. For example, the concept of object might encompass both simple and composite objects.

Next, we have to determine how the concept of composite object affects the way models are built. Generally, two models are of importance, the object model and the dynamic model. The object model defines the objects of the application and their categories (types). It includes a classification using subtyping of the categories of objects. The dynamic model defines how objects interact in terms of relationships and operation calls. In addition it may defines the semantics of the object's operations. In many existing OOADM, the process leading to the construction of these two models has many commonalities. This process can be described in a generic manner by the following activities:

(a) Identification of the objects and their properties
(b) Identification of the interactions among these objects
(c) Design through object decomposition and refinement
(d) Classification and factorization of objects properties
(e) Implementation of the object properties

The table below shows the correspondance between these activities and the steps of two existing OOADM, namely Object Modeling Technique (OMT) [Rumbaugh et al. 1991] and Booch's Object Oriented Design with Applications (OODA) [Booch 92]. On the one hand, OMT divides Analysis and Design in three parts: (1) Analysis consisting of building a model of the real-world situation starting with a problem statement, (2) System Design, the design of the overall architecture of the system, and (3) Object design which refines the object structure towards efficient implementation. Analysis is further subdivided in object

modeling, dynamic modeling and functional modeling activities. On the other hand, OODA is divided into four (non-sequential) major steps: (1) Identifying classes and objects at a certain level of abstraction, (2) Identifying the semantics of the objects and classes, (3) Identifying the relationships among classes and objects, and (4) Implementation of the classes and objects.

Activities	**Object Modeling Technique**	**Object Oriented Design with Applications**
(a)	Object modeling	- Identifying classes and objects at a certain level of abstraction - Identifying the semantics of the objects and classes
(b)	Dynamic modeling	Identifying the relationships among classes and objects
(c)	Object design	Identifying the relationships among classes and objects
(d)	Object design	Identifying the relationships among classes and objects
(e)	Object design	Implementation of the classes and objects

In the sequel, we provide the aspects of composition which are pertinent for each of the activity identified above.

Activity (a): Identification of the objects and their properties
Composite objects relevant to the application domain should be considered and their properties determined.

Activity (b): Identification of the interactions among these objects
In addition to the standard interactions among objects, we have to determine where some interactions may involve visible parts of known composite objects. [Configuration phase]

Activity (c): Design through object decomposition and refinement
In this activity, new objects may be required to describe properties of the known objects. These new objects can be considered as objects whose purpose is to provide properties

that are in relation with (in the sense of abstract implementation of) the properties of the object undergoing decomposition and refinement. Hence, these new objects should naturally be regarded as components of the object whose decomposition and refinement undercovered their existence. Along with this refinement is the allocation of responsibilities in terms of properties to the new objects and encapsulation of these new objects. [Juxtaposition and emergence phases]

Activity (d): Classification and factorization of objects properties

During this activity, type hierarchies are defined. Existing types may be reorganized to allow better factorization of the object properties. In particular, subtypes of composite types (i.e., composite types are types of composite objects) should be given special attention in order to cope with the behavior of the component types (i.e., component types are types of component objects). Subtyping in the context of composite objects is not yet well-defined. For instance, we may wonder whether a component type is allowed to be subtyped to define a new composite subtype?

Activity (e): Implementation of the object properties

Here, further details are given for the implementation of composite objects. Special attention should also be given to the implementation of composite object properties in terms of component object properties. [Juxtaposition phase]

It should be noted that during the integration of our composition framework within an existing OOADM, it is not mandatory to take the whole framework. The framework is adaptable to various levels (degrees) of composition. A version including only the first step, called configuration, can be applied to the description of object frameworks. In certain situations, only the identity of the whole is of importance. This is the case when one focuses on the description of the structure of objects, like in most database applications, where only the steps of configuration and juxtaposition are required. Finally, a full fledged version including all the steps is needed when emergent properties are taken into account.

4. Conclusion

This paper reviews the theme of composition of objects. It has been noted that there are multiple interpretations of the term *object composition*. A significant challenge in research on object composition is to provide a non-ambiguous and workable definition of composition suitable for handling the most important aspects. The conceptual framework presented in this paper takes a step towards this goal by providing a model for composite objects and a process for the composition of objects. The main points of our framework are twofold:

(1) A definition and motivation using Ontology of composite objects: We view composite objects as objects resulting from the superposition of three specific kinds of properties: *inherent*, *aggregate* and *emergent* properties. In existing approaches to composition, this distinction is ignored. We show that this distinction clarifies the role that is played by each component object by explicitly defining how the properties of the component objects contribute to the properties of the composite object.

(2) A proposal for a composition process which enforces separation of concerns: Composite objects are built through superposition of properties. In order to achieve this principle, a stepwise process to composition is recommended such that each step is built upon others. A given step is concerned with related aspects of a composite object. This allows focusing on a specific aspect of composition, for instance the interactions among components. The composition process has three steps: *configuration*, *juxtaposition* and *emergence*. Configuration defines the interactions among the components, i. e., the internal architecture of the composite. Juxtaposition determines the relationship which links the composite to its components and the resulting properties, i.e., the implementation of the composite in terms of components. This step defines the inherent and aggregate properties of a composite. The last step, emergence is aimed at describing the emergent properties. It is made analogous to specialization through the definition of additional properties.

We have also outlined how this framework can be integrated into existing OOADM. The integration into a specific OOADM requires further analysis and left for future development. Other aspects which are candidates for future developments are: (i) formalizing the conceptual framework, and (ii) applying the conceptual framework to sizable examples. In particular, we would like to experiment this approach in application

domains such as enterprise modeling, office automation, multimedia objects, telecommunications systems (particularly for the handling of feature interactions and system management), CAD, CASE, and CAM. In these application domains, we need to consider both structural and behavioral aspects of composite objects.

Annex: Description of composite objects using the template

Cab composite object specification:

cab composition
- **configuration**
  - *roles:*
    - Load, Cab-doors
  - *interactions among roles:*
    - -- none
- **juxtaposition**
  - Load
    - is-part-of characteristics:
      - exclusive, independent, extensive
    - functional role of the component:
      - -- none
  - Cab-doors
    - is-part-of characteristics:
      - exclusive, independent, extensive
    - functional role of the component:
      - -- used to control the access to the cab
  - *inherent properties:*
    - door-open
      - property description:
        - -- operation conveyed from Cab-door
        - -- also related to door-close property
      - phenomena of interest captured by this property:
        - -- controls the access to the cab
    - door-close
      - property description:
        - -- operation conveyed from Cab-door
        - -- also related to door-open property
      - phenomena of interest captured by this property:
        - -- controls the access to the cab
  - *aggregate properties:*
    - weight
      - property description:
        - -- physical property of the cab
      - phenomena of interest captured by this property:
        - -- this property is the sum of the corresponding component weights
    - position
      - property description:
        - -- physical property of the cab
      - phenomena of interest captured by this property:
        - -- this property is the same for both composite and components
    - availableVolume
      - property description:
        - -- physical property of the cab
      - phenomena of interest captured by this property:
        - -- this property determines the possibility of loading the cab with additional goods

end cab composition

Floor composite object specification:

floor composition
- **configuration**
  - *roles:*
    - Floor-doors, Floor-buttons
  - *interactions among roles:*
    - controls(Floor-doors, Floor-buttons)
      - phenomena of interest captured by this relationship:
        - -- doors are controlled through floor buttons
- **juxtaposition**
  - Floor-doors
    - is-part-of characteristics:
      - exclusive, independent, extensive
    - functional role of the component:
      - -- used to control the access to the cab
  - Floor-buttons
    - is-part-of characteristics:
      - exclusive, independent, extensive
    - functional role of the component:
      - -- used to control the cab
  - *inherent properties:*
    - door-open
      - property description:
        - -- operation conveyed from Floor-door
        - -- also related to door-close property
      - phenomena of interest captured by this property:
        - -- controls the access to the cab
    - door-close
      - property description:
        - -- operation conveyed from Floor-door
        - -- also related to door-open property
      - phenomena of interest captured by this property:
        - -- controls the access to the cab
    - button-push
      - property description:
        - -- operation conveyed from Floor-button
      - phenomena of interest captured by this property:
        - -- triggers a specific operation
- **emergence**
  - *emergent properties:*
    - Floor-number
      - property description:
        - -- uniquely identify a floor

end floor composition

Shaft composite object specification:

shaft composition

configuration

roles:

CounterWeight, Motor, Cab

interactions among roles:

tractorOf(CounterWeight, Motor, Cab)

phenomena of interest captured by this relationship:

-- the motor serves to move the cab

associated constraints:

-- there exists a relation between the positions of cab and CounterWeight

juxtaposition

CounterWeight

is-part-of characteristics:

exclusive, independent, non-extensive

functional role of the component:

-- used to balance with the cab

Cab

is-part-of characteristics:

exclusive, independent, non-extensive

functional role of the component:

-- used to move goods

Motor

is-part-of characteristics:

exclusive, independent, non-extensive

functional role of the component:

-- used to move upward or downward the cab

inherent properties:

door-open

property description:

-- operation conveyed from the cab, also related to door-close property

phenomena of interest captured by this property:

-- controls the access to the cab

door-close

property description:

-- operation conveyed from the cab, also related to door-open property

phenomena of interest captured by this property:

-- controls the access to the cab

aggregate properties:

weight

property description:

-- physical property of the shaft

phenomena of interest captured by this property:

-- this property is the sum of the corresponding component properties

emergence

emergent properties:

overloadWarning

property description:

-- signal of overload of the cab

phenomena of interest captured by this property:

-- the shaft has a limited capacity

meanServiceTime

property description:

-- characterizes the availability of the cab

phenomena of interest captured by this property:

-- waiting time for the user

end shaft composition

References

[Abadi, Lamport 93]
Abadi, M., Lamport, L., *Composing Specifications*, ACM Transactions on Programming Languages and Systems, Vol. 15, No. 1, 73-132, 1993.

[Ablowitz 39]
Ablowitz, R., *The Theory of Emergence*, Philosophy of Science, Vol. 6, No. 1, 1-16, 1939.

[Angyal 39]
Angyal, A., *The Structure of Wholes*, Philosophy of Science, Vol. 6, No. 4, 26-47, 1939.

[Bapat 94]
Bapat, S., Object Oriented Networks Models for Architecture, Operation and Management, Prentice-Hall, 1994.

[Bochmann et al. 92]
Bochmann et al., The IGLOO Project: Research Proposal -- Technical Description, CRIM, May 1992.

[Booch 92]
Booch, G., Object-Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Co. Inc., 1992.

[Brooks 87]
Brooks, F., *No Silver Bullet: Essence and Accidents of Software Engineering*, IEEE COMPUTER, Vol. 20, No. 4, 10-20, 1987.

[Bunge 77]
Bunge, M., *Treatise on Basic Philosophy: Ontology I: The Furniture of the World*, Reidel, 1977.

[Bunge 79]
Bunge, M., *Treatise on Basic Philosophy: Ontology II: The World of System*, Reidel, 1979.

[Cameron, Velthuijsen 93]
Cameron, J., Velthuijsen, H., *Feature Interactions in Telecommunications Systems*, IEEE Communications Magazine, August 1993, pp. 46-51.

[Cargill 91]
Cargill, T., *A Case Against Multiple Inheritance in C++*, Proceedings of USENIX Conference, 1991.

[Cargill 92]
Cargill, T., C++ Programming Style, Addison-Wesley, 1992.

[Champeaux 91]
de Champeaux, D., *Object-Oriented Analysis and Top-down Software Development*, ECOOP'91, pp. 360-376.

[Civello 93]
Civello, F., *Roles for composite objects in object-oriented analysis and design*, OOPSLA'93, pp. 376-393 , 1993.

[Coad 92]
Coad, P., *Object-Oriented Patterns*, CACM Vol. 35 No 9, 1992.

[Coleman et al. 94]
Coleman et al., Object-oriented Development, THE FUSION METHOD, Prentice-Hall, 1994.

[Forman 87]
Forman, I., On the design of large distributed systems, Microelectronics and Computer Technology Corporation, Report STP-098-86, January 1987.

[Halper et al. 92]
Halper et al., *An OODB "Part" Relationship Model,* Proceedings of the ISMM International Conference on Information and Knowledge Management CIKM-92, Baltimore MD-USA, November 1992.

[Helm et al. 90]
Helm, R., Holland, I., Gangopadhyay, D., *Contracts: Specifying Behavioral Compositions in Object-Oriented Systems*, ACM SIGPLAN Notices, Vol. 25, pp. 169-180, 1990.

[IWSSD-4]
Proceedings of the Fourth International Workshop on Software Specification and Design, Monterey, 1987.

[Johnson, Opdyke 93]
Johnson, R., Opdyke, W., *Refactoring and Aggregation*, in Proceedings of Object Technologies for Advanced Software, Nishio, S. and Yonezawa, A. (Eds.) LNCS 742, pp. 264-278, November 1993.

[KS 93]
Kurki-Suonio, R., *Stepwise Design of Real-Time Systems*, IEEE Transactions on Software Engineering, Vol. 19, No. 1, pp. 56-69, 1993.

[Lam, Shankar 92]

Lam, S., Shankar, A., *Specifying modules to satisfy interfaces: a state transition system approach*, Distributed Computing (1992) 6:39-63.

[Lam, Shankar 94]
Lam, S., Shankar, A., *A Theory of Interfaces and Modules. I -- Composition Theorem*, IEEE TSE Vol. 20, No. 1, January 1994, pp. 55-71.

[Lamport 92]
Lamport, L., *Critique of the Lake Arrowhead Three*, Distributed Computing (1992) 6:65-71.

[Lee et al. 93]
Lee, P., Chen, D., Ku, K.-L., *A modeling approach to the construction of object-oriented operating systems*, JOOP, Vol. 6, No. 7, pp. 52-63, 1993.

[Liu 92]
Liu, L., *Exploring semantics in aggregation hierarchies for object-oriented databases*, IEEE Conference on Data Engineering, pp. 116-125, 1992.

[Odell 94]
Odell, J., *Six different kinds of Composition*, JOOP Vol. 5, No. 8, January 1994, pp. 10-15.

[ODP 93]
ISO, *Reference Model for Open Distributed Processing*, ISO/IEC JTC1/SC21/WG7, 1993.

[Pressman 93]
Pressman, L., Software Engineering. A Practitionner Approach, second edition, McGraw-Hill, Inc. 1993.

[Ramazani, Bochmann 93]
Ramazani, D., Bochmann, G.v., *Understanding behavioral compositions through is-part-of relationships*, working paper, June 1993.

[Reenskaug et al. 92]
Reenskaug, T. et al. *OORASS: seamless support for the creation and maintenance of object oriented systems*, JOOP, Vol. 5, No. 6, pp. 27-41, October 1992.

[Robinson 92]
Robinson, P., Object-oriented Design, Chapman & Hall, 1992.

[Rumbaugh et al. 1991]
Rumbaugh, J. et al. Object-Oriented Modeling and Design, Prentice Hall, Englewood Cliffs, 1991.

[Rumbaugh 93]

Rumbaugh, J., *Disinherited! Examples of misuse of inheritance*, JOOP Vol. 5, No. 9, pp. 22-24, February 1993.

[Sakkinen 89]
Sakkinen, M., *Disciplined Inheritance*, Proceedings of ECOOP Conference, 1989, pp. 39-56.

[Spivey 89]
Spivey, J.M. The Z Notation: A Reference Manual, Prentice-Hall, Englewood Cliffs, N.J., 1989.

[Wand 88]
Wand, Y. *A proposal for formal object model*, in W. KIM, F.H. Lochovsky, Eds, Object-Oriented Concepts, Databases, and Applications, Addison-Wesley, Reading, Massachusets, 1988.

[Winston et al. 87]
Winston, M., Chaffin, R., Hermann, D., *A taxonomy of part-whole relation*, Cognitive Scince 11:417-444, 1987.

[Yemeni 93]
Yemeni, Y., *A Critical survey of network management protocol standards*, working paper, January 1993.

[Tversky, Hemenway 84]
Tversky, B., Hemenway, K., *Objects, Parts, and Categories*, Journal of experimental Psychology: General, Vol. 113, No. 2 pp. 169-191, 1984.

[Zave, Jackson 93]
Zave, P., Jackson, M., *Conjunction as composition*, ACM Transactions on Software Engineering Methodology, Vol. 2, No.4 , October 1993.