Specification of composite objects based on the ODP Reference Model

D. Ramazani and G. v. Bochmann
Département d'informatique et de recherche opérationnelle
Université de Montréal
C.P. 6128, Succursalle Centre-Ville
Montréal, Canada H3C 3J7
Phone: (514) 343-7484, fax: (514) 343-5834, e-mails: {bochmann, ramazani}@iro.umontreal.ca

Abstract

In this paper, we describe our experience in using the RM-ODP to specify composite objects. The concept of a composite object as defined in RM-ODP does not take into account its dynamic structure as well as the classification of its properties into inherent, aggregate and emergent properties. To make this possible, we have to complement the description of composite objects with explicit contracts involving the composite object and its components.

This interpretation of composite objects in terms of ODP concepts is technically sound. It allows a clean definition of the structure, the inherent and aggregate properties of the composite object. However, this approach is conceptually questionable since its guiding philosophy is based on ignoring the distinction between composition and interconnection of objects. This observation is grounded on the usage of contracts for defining significant aspects of object composition. We come out with the same conclusion when experimenting the description of composite objects using Darwin. What these experiments show is that we still have to rely on the concept of interconnection of objects to define the semantics of composition of objects.

Keywords

Composition of objects, Darwin, Distributed Systems, ODP

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

1 INTRODUCTION

Motivation of this work

The Reference Model for Open Distributed Processing (RM-ODP) is based on object-oriented concepts (ISO-1, ISO-2, ISO-3, ISO-4, 1995). Among these concepts, object composition plays a key role in the specification of ODP systems. Object composition consists of describing some objects of a system in terms of other objects of this system. The former objects are called composite objects, while the latters are component objects. RM-ODP goes further in the usage of object composition by assuming that any ODP system is a composite object. It proposes the application of ODP concepts to specify this object. In that sense, the concept of object composition is used for structuring ODP systems.

With these considerations in mind, it is fair to assume that the RM-ODP is adequate for describing composite objects. This assumption is also supported by the fact that a composite object is distributed among its components, i.e. composite objects are analogous to distributed systems. In addition, a composite object is a distributed system which needs to be open since it has to interact with other objects or to be composed with other objects to form more complex objects. At this point, based on theoretical and practical reasons, it is interesting to assess how RM-ODP can be used to specify composite objects. From the theoretical point of view, the analogy between composite objects and distributed systems makes composite objects a good starting point for assessing the adequacy and accuracy of RM-ODP for the specification of distributed systems. On the other hand, in practice, complex applications in engineering and telecommunications often imply composite objects (Ramazani and Bochmann, 1995). These applications will gain to be specified using RM-ODP. The uniform usage of RM-ODP concepts at any level of he description of these applications is recommended.

Summary of our experience

In this paper, we describe our experience in using the RM-ODP to specify composite objects. The concept of object composition used in this work originates from (Ramazani and Bochmann, 1995). In (Ramazani and Bochmann, 1995), composite objects are viewed from a new perspective which focuses on the linkage between the structure and the behavior of these objects. Composite objects contrasting with simple objects have three kinds of properties, namely inherent, aggregate and emergent properties. In addition, they have a structure in terms of components and interconnections among these components. This is further explained later in this paper.

The concept of a composite object as defined by RM-ODP does not take into account its dynamic structure as well as the classification of its properties into inherent, aggregate and emergent properties. To make this possible, we have to complement the description of composite objects with contracts, using RM-ODP parlance, aimed at defining the structure of these objects and allowing the description of inherent and aggregate properties. The structure of a composite object is defined as a contract between its components. Inherent properties are specified as properties of the composite object. The semantics of these properties is defined by contracts between the composite object and some of its components. Aggregate properties are specified analogously, except that the contracts relate the composite object and all its components. The definition of emergent properties does not require the usage of contracts.

This interpretation of composite objects in terms of ODP concepts is technically sound. It allows a clean definition of the structure, the inherent and aggregate properties of the composite object. However, this approach is conceptually questionable since its guiding philosophy is based on using the concept of interconnections of objects to define the semantics of object composition. This observation is grounded on the usage of contracts for defining significant aspects of object composition. A contract is aimed at describing static and dynamic interactions between objects.

Fortunately, this observation is not specific to RM-ODP. We come out with the same conclusion when experimenting the description of composite objects using Darwin (Kramer et al, 1992), a specification language for distributed systems. Darwin is one of the most popular configuration language for the specification of distributed systems using an object-oriented approach. In Darwin, the concepts of (1) composite component and (2) binding are conjointly used to define composite objects. Bindings are useful for describing the structure, the inherent and aggregate properties of composite objects. It appears that the bindings of Darwin are a way of representing interconnections between objects. Therefore, they can be used to define the structure as well as inherent and aggregate properties of a composite object. Although the bindings of Darwin and the contracts of ODP can both be used for defining interconnections between objects, ODP contracts are more expressive than bindings. A binding allows two or more objects to synchronize through the properties which are mapped. The constraints related to the synchronization are distributed among the objects which are bound. In an ODP contract, in addition to the synchronization of objects, the constraints related to the interaction of these objects are localized within the contract, and we may define some (local) properties which are conjointly used by the objects which participate in the contract. In other words, a binding specification can be replaced by a contract specification while the converse is not necessarily true.

What these experiments show is that we still have to rely on the concept of interconnection of objects to define the semantics of composition of objects. This observation relegates us to the debate on the distinction between composition and interconnection of objects. We believe that such a distinction is necessary in the context of complex distributed systems including ODP systems. In addition, we propose that the linkage between structure and behavior of composite objects has to be taken into account within RM-ODP.

Organization of this paper

The remainder of this paper is structured as follows. In Section 2, we introduce the concept of composite objects. Next, we show the analogy between composite objects and distributed systems. This is followed in Section 3 by the approach which is taken for specifying composite objects using the RM-ODP. We also discuss advantages and shortcomings of the approach. To illustrate this approach, we use a small application. The notation used for formulating ODP specifications is a combination of OMT (Rumbaugh, 1991) and ad hoc texts. After this exercise in specification, in Section 4, we explain how the same example can be specified using Darwin. Lessons learned from the two experiments are compared. From this comparison results a distinction between composition and interconnection of objects. To close this paper, we point out future directions of this work.

2 COMPOSITE OBJECTS

2.1 The concept

Existing object-oriented methodologies propose many approaches to the specification of composite objects. These approaches can be classified into four categories as described in (Ramazani and Bochmann, 1995). For example, the Fusion method (Coleman et al, 1994) represents composition as the abstraction of a given relationship among component objects. This way of handling composition shows the interconnections between the component objects. It focuses on the structure and neglects the behavior of the composite objects. In OMT (Rumbaugh, 1991), composition of objects is described by means of is-part-of (aggregation) relationships relating the composite object to its components. This approach has the advantage of describing, in an hierarchical manner, the structure of an object. In addition, it captures the fact that the composite object has access to its components. In methods such as Booch's OODA (Booch, 1994) and HOOD (Robinson, 1992), composition of objects is represented by attribution, i.e., component objects are represented by attributes of the composite object. Attribution captures the hierarchical organization of composite objects. However, it precludes any distinction between composition and interconnections of objects. There are situations where composition of objects is modeled by multiple inheritance. Such situations are reported in (Cargill, 1991), (Rumbaugh, 1993) and (Sakkinen, 1989). 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 problems when reusing such a specification.

All these approaches distinguish themselves by focusing on some aspects and neglecting other aspects of composition. Among these aspects, the contribution of the structure of the composite object to its behavior is ignored. To alleviate this problem, we propose in (Ramazani and Bochmann, 1995) a new perspective on composition of objects. In the sequel, we briefly present this approach.

Definition 1: *Composite object*

A composite object is an object with an internal structure. It has three kinds of properties, namely inherent, aggregate and emergent properties. A property may denote an attribute, an operation, a behavior, a structural relationship, a behavioral interaction or a sequence of interactions. The structure of a composite object consists of:

- the components (type and number of instances within the composite object);
- the interconnections and/or dynamic interactions between the components of the composite object.

Definition 2: *Inherent property*

An inherent property is a property of the composite object such that the semantics of this property is defined by some component of this composite object. When the component on which depends this property is absent in the composite object, the inherent property is undefined.

Definition 3: *Aggregate property*

An aggregate property represents the combination of the properties of all components. The aggregation mechanisms used for combining these properties are defined at the composite object

level and they depend on the way the components are interconnected.

Definition 4: *Emergent property*
An emergent property is a property of the composite object which does not directly depend upon the properties of the components.

What really makes a composite object 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.

To illustrate the properties of a composite object, consider a door. It may be a component of an house. It is fixed on a frame which is also a component of the house. It can be used by a person. We view the door as a composite object consisting of a prefabricated section, a handle, a pair of hinges and a lock. The lock and the handle are installed on the prefabricated section. The hinges are fixed on the prefabricated section. They are also fixed on the frame of the house in order to hold the door.

Among the inherent properties of a door, we find its attributes size and color. These attributes are determined by the prefabricated section, i.e. the size and the color of the prefabricated section are also respectively the size and the color of the door. When the prefabricated section is substituted by another prefabricated section, the door acquires the color and the size of the new prefabricated section. The fact that the hinges are fixed on the frame implies that the door on which these hinges are fixed is also fixed on the same frame. We may say that "fixed on" is an inherent relationship of the door. There are also inherent operations such as lock and unlock which are offered by the lock component.

Two major aggregate properties can be defined for the door, its weight and moving operation. Its weight results from the aggregation by summation of its component weights. Moving the door causes its components to move. Therefore, moving the door is the concurrent aggregation of moving operations of its components. Closing and opening operations of the door are considered as two distinct refinements of the move operation. Emergent properties of the door include its price and its inclusion within the house as a component.

2.2 Analogy between composite objects and distributed systems

In the context of the RM-ODP (ISO-1, 1995), distributed systems are characterized by the following properties.

Remoteness: Components of a distributed system may be spread across space.

Concurrency: Any component of a distributed system can execute in parallel with any other components.

Lack of global state: The global state of a distributed system may be indeterminate.

Partial failures: Any component of a distributed system may fail independently of any other components.

Asynchrony: Related changes in a distributed system cannot be assumed to take place at a single instant.

It is interesting to note that these properties apply for composite objects with respect to their components. To illustrate this point, we consider a portable cassette player. It is a composite object which consists of a case and an earphone. The case is located on a table while the earphone is on the ground. Using the portable cassette player, we may interpret properties of distributed systems in the context of composite objects as follows.

Remoteness: Components of a composite object may be spread across space. For instance, consider the case and the earphone, these components are not at the same location. One is on the table while the other is one the ground.

Concurrency: Any component of a composite object can execute in parallel with any other components. As an example, while the cassette is moving forward in the case due to the action of the motor, the earphone delivers some sound to the environment. These functions constitute concurrent activities of distinct components.

Lack of global state: The global state of a composite object depends upon the state of its components. For instance, the state of the portable cassette player is a combination of the state of the case and the state of the earphone. Further, considering the state of the case, it consists of the combination of the state of its components such as the motor, the amplifier, the head, etc.

Partial failures: Any component of a composite object may fail independently of any other components. Consider for example that the earphone is scrapped using an hammer. This action does not preclude the playing of the cassette in the case.

Asynchrony: Related changes in a composite object cannot be assumed to take place at a single instant. For instance, as soon as the cassette is playing within the case, the earphone will deliver the sound. This requirement on the behavior of the portable cassette player does not imply that these events take place at a single instant.

3 INTERPRETATION OF COMPOSITE OBJECTS USING RM-ODP

3.1 The approach

In Section 2, we have described composite objects as objects being characterized by its (1) structure in terms of components and interconnections among these components, and (2) by its inherent, aggregate and emergent properties. According to RM-ODP, a composite object is an object compliant to the object model. RM-ODP recognizes the existence of a static structure for composite objects in terms of components and static interconnections among these components since it advocates the usage of configurations of components to describe this static structure. In addition, RM-ODP says nothing about inherent, aggregate and emergent properties for composite objects.

In order to take these aspects into consideration, we propose to complement the definition of the static structure of composite objects by contracts describing the role of each component as well as the requirements in terms of interconnections and interactions among these components

(dynamic structure). Within RM-ODP, the concept of contract characterizes and regulates the cooperation of objects. A contract is an agreement that governs the cooperation among a number of objects, and it embodies the ideas of obligations and expectations associated with cooperating objects. In this study, we restrict the concept of contract to the roles of objects and the obligations required for these roles. The concepts of quality of service and kind of behavior invalidating the contract are ignored. These restrictions have no impact upon the results of this experiment.

The structure of the composite object is defined by a contract between its components. We define inherent properties of composite objects by first defining these properties as properties of the composite since a composite object is an object compliant to the object model, then by specifying a contract including the composite and the component which provides this property. The obligation embedded in this contract consists of defining the inherent property of the composite in terms of some property of the component.

The representation of aggregate properties is similar to that of inherent properties, except that the contract includes the composite and all its components. The obligations of this contract specify the semantics of aggregation mechanisms which are used for combining the component properties. Emergent properties, since they have no direct relation with the component properties, are defined as usual properties of objects.

As it shall be exemplified later, the specification of a composite object requires one or many contracts for its structure, one or many contracts for its inherent properties, and one or many contracts for its aggregate properties. These contracts define the semantics of interactions between (1) the components, and (2) the components and the composite. According to RM-ODP, objects and composite objects are specified by class templates. In our approach, in addition to the class templates, we use a set of contracts for composite objects. These contracts are definitional, i.e. they contribute to the definition of the composite object.

This interpretation of the concept of composite object using two concepts of RM-ODP is not regarded as unique. However, to the best of our knowledge, it is optimal since it uses a minimum number of RM-ODP concepts and takes advantage of the semantics of these concepts. In addition, the approach takes care that an ODP specification has plausible intuitive interpretation. Besides all these advantages, this approach has disadvantages: (1) the definition of a composite object is spread across many pieces of the specification. (2) The specifications are verbose. (3) When it comes times to relate the different pieces of a composite object specification, care must be taken to avoid confusion. However, this is the price we have to pay to have accurate and precise specifications of composite objects in the context of RM-ODP.

3.2 An example

An ODP system is anything including some peripheral device. Due to lack of space, we consider in this paper a simpler example, namely the specification of a printer in terms of ODP concepts. The printer is described as a composite object. This example is a simplification and incomplete description of a real laser printer. However, it provides interesting insights into the use of RM-ODP for the specification of composite objects.

A laser printer is a peripheral device used for printing files. Printing consists of transcribing the output of a computer in the correct layout onto paper. The printer is connected to a host computer which provides data and control signals. For sake of brevity, we consider the following components of the printer:

• the paper tray which holds the paper;

• the paper control circuit which monitors the paper tray;
• the electrophotographic cartridge (EP cartridge) which contains the toner;
• the control panel which allows the user to maintain the printer by selecting appropriate printing options, operating modes as well as allowing the functions of turning the printer on and off;
• the controller, which is the heart of the printer. It has circuitry that operates the printer and it is responsible for controlling the paper control circuit, the EP cartridge and the control panel. In addition, the controller has a memory of some given capacity.

The RM-ODP object model is fairly general and it makes a minimum number of assumptions. Therefore, we may use the notation of OMT to represent ODP specifications, as advocated in (ISO-1, 1995). However, there is no corresponding concept of OMT for representing ODP contracts. To overcome this shortcoming, we use an ad hoc textual convention for representing ODP contracts using the OMT notation. This convention is illustrated in Figure 1. In this ad hoc notation, the utility of a contract can be twofold. It can be used to specify the semantics of interactions between objects. In this case <utility> is the name of the association abstracting these interactions. It can be used to define the properties of a composite object in terms of properties of its components. In this latter case, <utility> is "composition".

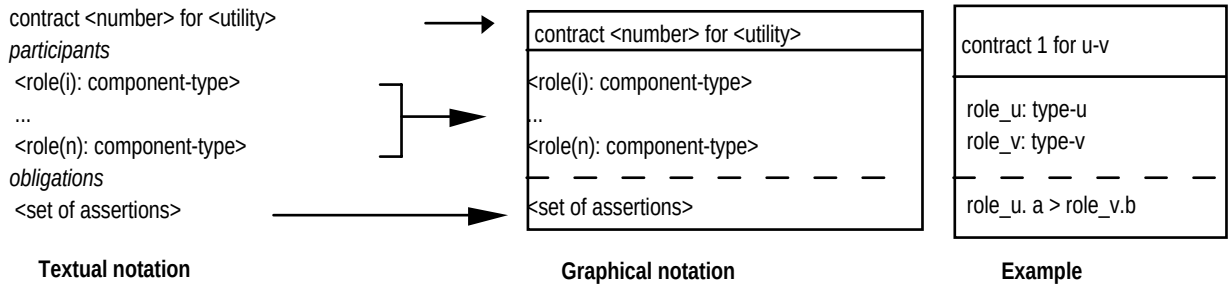

Figure 1 RM-ODP Contracts in OMT.

In a contract, an assertion can be any predicate allowed in OMT specifications as well as a behavior predicate involving operators like ≡ for the definition of an operation in terms of other operations or || for parallel execution of operations.

The printer, as a composite object, can be described in terms of its structure, inherent, aggregate and emergent properties as follows (see Figure 2).

Printer

Structure

Components

paper tray, paper control circuit, controller, EPcartridge, control panel

Interconnections and interactions among components

- the paper control circuit monitors the status of the paper tray;
- the controller operates the paper control circuit;
- the controller monitors the level of toner of the EPcartridge;
- the controller controls the control panel in the sense that each function offered by the control panel is in fact executed by the controller. The control panel acts as an interface of the controller to the user for management purposes.

Inherent properties	*[From]*
memory	[controller]
supplytraystatus	[controller]
leveloftoner	[controller]
printoptions	[controller]
operatingmodes	[controller]
set-printoptions	[controlpanel]
set-operatingmodes	[controlpanel]
set-printoptions	[controlpanel]
set-operatingmodes	[controlpanel]
turn-on	[controlpanel]
turn-off	[controlpanel]
print(file, options)	[controller]

Aggregate properties	*[Aggregation mechanism]*
location	[spatial aggregation]
weight	[summing]
move(newlocation)	[parallel execution]

Emergent properties

printspeed
resolution
identification
remove-papertray
insert-papertray
remove-cartridge
insertcartridge

Figure 2 The laser printer as a composite object.

There is another composite object within the printer, namely the controller which has a memory as a component. The controller is described as follows (see Figure 3).

Controller

Structure	
Components	
memory	
Inherent properties	*[From]*
memorycapacity	[memory]
Aggregate properties	*[Aggregation mechanism]*
location	[spatial aggregation]
weight	[summing]
move(newlocation)	[parallel execution]
Emergent properties	
supplytraystatus	
leveloftoner	
printoptions	
operatingmodes	
change-printoptions	
change-operatingmodes	
turn-on-the-printer	
turn-off-the-printer	
print(file, options)	

Figure 3 The controller as a composite object.

These composite objects are interpreted using RM-ODP concepts as described in the sequel. The specification of the information viewpoint of the printer is expressed using an OMT object model as shown in Figure 4. The object model shows the classes and their relationships to each other. Classes are described in terms of operation signatures and attributes. Among the relationships between classes, the object model portrays composition of objects (aggregation relationships), object configurations in terms of associations between objects, and class and subclass hierarchies (generalization relationships).

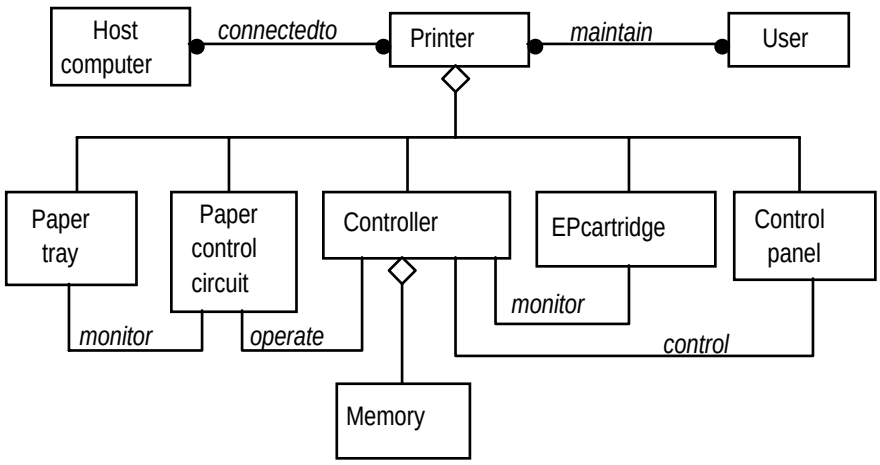

Figure 4 The printer information viewpoint: Composition and interconnection of objects.

In Figure 4, we focus on the general structure of the printer in terms of composition and

interconnections among its components. Composition is indicated by aggregation relationships. Aggregation is indicated in the OMT notation by placing a diamond at one end of the line relating two classes. The diamond is attached to the composite object class. Interconnections are indicated by associations between classes. An association is drawn as a line between two classes. As an example, in Figure 4, the printer consists of a certain number of components like the controller and the paper control circuit to name few of them. This is indicated by an aggregation relationship relating the printer to the paper control circuit and the controller. In addition, these two components are interconnected. This is indicated by an association represented as a line between paper control circuit and controller. The line is labeled with the name of the association, which is "operate" in this case. The multiplicity of associations is indicated with no marker at the end of the line for a multiplicity of 1, a hollow ball for a multiplicity of 0 or 1 and a solid ball for a multiplicity of 0 or many. In Figure 4, the multiplicity of paper control circuit in the association "operate" is 1 and also 1 for the controller. This means that only one paper control circuit may be operated by only one controller. In the association "maintain", the multiplicity is many for user and also many for printer. This means that the user may maintain many printers and each printer can be maintained by many users.

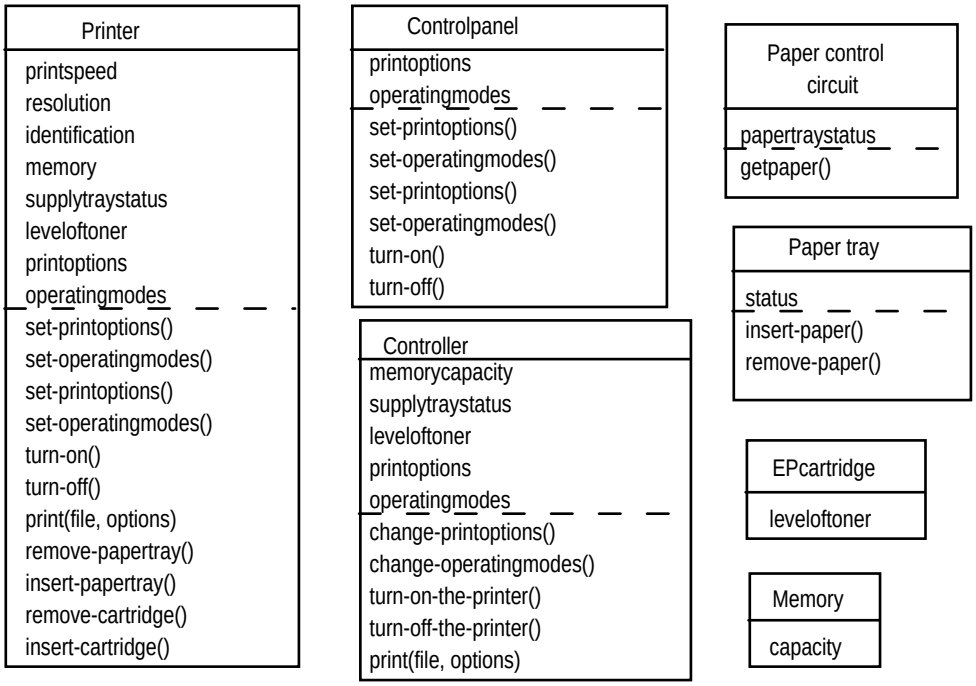

Figure 5 The printer information viewpoint: Class description.

In OMT, a class is drawn as a 3-part box, with the class name in the top part, a list of attributes (with optional types) in the middle part, and a list of operations (with optional argument lists and return types) in the bottom part. For instance, the paper control circuit class as pictured in Figure 5 has an attribute papertraystatus and an operation getpaper(). After having described each class in terms of attributes and operations, these can be omitted to reduce detail on other parts of the object model. In addition all the classes are subclasses of Physical object. A Physical object has a location, a weight and it can be moved from one location to another.

In the sequel, the computational viewpoint is described in terms of contracts as illustrated in

the Figures 6, 7 and 8. The contracts which define the structure of the printer in terms of interactions between its components are numbered 1, 2, 3 and 5. The contract 4 defines the semantics of the inherent property memorycapacity of the controller.

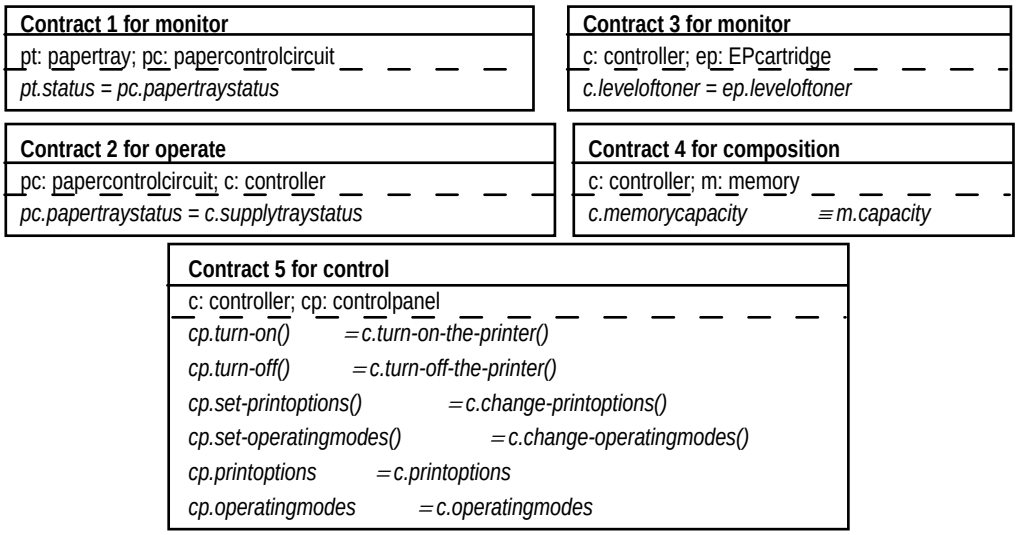

Figure 6 The contracts 1, 2, 3, 4 and 5.

The contract 6 and 7 define in turn the semantics of the inherent properties of the composite object printer. Finally, the contract 8 defines the aggregation mechanisms used for determining the semantics of aggregate properties of the printer. Location is defined as a function **f** of printer component locations. Weight is defined by summation of component weights and move by parallel execution of corresponding component operations. Note that in a contract when the names of the properties coincide, this does not means that these properties have to be conjoined. There is a distinction between a ≡ b and a = b. The first is an assertion stating that a is defined in terms of b. It is like aliasing in programming languages. The latter assertion equates a and b. It is like stating that a ^{3} b and b ^{3} a which is quite different from aliasing. In addition, the first assertion a ≡ b states that a can not exist without b, while the latter assertion distinguishes the existence of a from that of b. It imposes a constraint on their respective values.

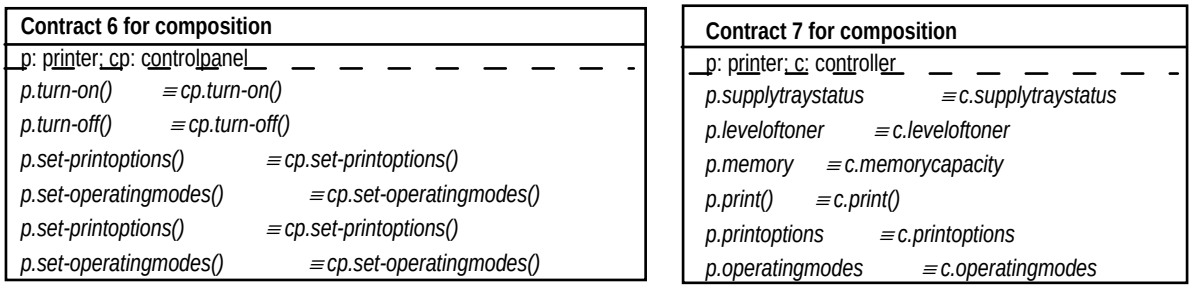

Figure 7 The contracts 6 and 7 define the semantics of the inherent properties.

For example, (1) p.turn-on() ≡ cp.turn-on() and (2) cp.turn-on() = c.turn-on-the-printer() do not have the same meaning. In (1), the property turn-on() for the printer is defined in terms of the property turn-on() of the control panel. The effects of these properties are the same and the

property turn-on() of the printer is undefined when the control panel is absent. In (2), the property turn-on() for the control panel is simply equated to the property turn-on-the-printer() of the controller.

Contract 8 for composition

p: printer; pt: papertray; pc: papercontrolcircuit; c: controller; ep: EPcartridge; cp: controlpanel

p.location ≡ *f(pt.location, pc.location, c.location, ep.location, cp.location)*

p.weight ≡ *(pt.weight, pc.weight, c.weight, ep.weight, cp.weight)*

p.move() ≡ *pt.move()|| pc.move()|| c.move()|| ep.move()||cp.move()*

Figure 8 The contract 8 defines the semantics of the aggregate properties of the printer.

4 INTERPRETATION OF COMPOSITE OBJECTS USING DARWIN

Darwin is a specification language for distributed systems (Kramer et al, 1992). It is based on the definition of the program structure as a set of active objects (components) with explicit interfaces and bindings. The behavior of an object is defined in terms of services it provides to its environment and services it requires from its environment. The set of provided and required services constitutes the interface of the object. Interconnections between objects consist of bindings between required and provided interfaces. A binding matches a provided service to a required service of distinct objects. The bindings are type-checked.

Darwin is also based on the use of hierarchical composition to define composite objects. Composite objects are defined analogously to simple objects, except that the provided services (respectively the required services) of the composite object are bound to provided services (respectively required services) of its components.

In order to make the presentation concise, we made some simplifications to the printer example. We consider that the printer consists of only two components, the controller and the control panel. It has two inherent properties, memory and turn-on; two aggregate properties weight and move; and one emergent property, resolution. In spite of these simplifications, the example still provides sufficient insights in the interpretation of composite objects in terms of Darwin concepts.

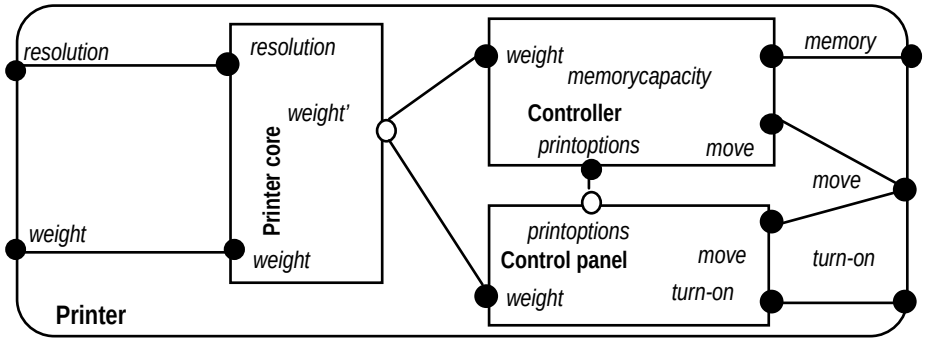

Figure 9 Darwin specification of the printer.

As illustrated in Figure 9, within Darwin specifications, a component is represented by a rectangle. Its provided services are represented by solid balls while the required services are

represented by hollow balls. Bindings between component services are indicated by lines linking the bound services. Component composition is graphically represented by rectangle inclusion.

The structure of a composite object is defined by rectangle inclusion of its components and possibly bindings between the components when these interact. Inherent properties are represented by provided services (respectively required services) of the composite which are bound to component services. For example, the printer memory and the function turn-on are inherent properties since their are bound to component services.

Aggregate properties can be represented in many ways. For instance, the function move which requires the moving of all the components can be represented as the parallel execution of corresponding component functions. This is represented by binding the move service of the composite object to move services of the controller and the control panel. However, in the case of the aggregate property weight, as this property requires the summation of corresponding component weights, we need to define at which level this summation shall be executed. In Darwin, all the services provided (respectively services required) by a composite object have to be implemented in terms of component services. This means that the summing of the weights have to be done at the component level. This requires the introduction of a dummy component, here called printer core. The dummy component is responsible for collecting the weights of the other components and summing these weights in order to obtain the overall weight of the printer. As a consequence, the dummy component provides the service "weight" to the printer. This looks like an implementation bias, but we have no choice since the composite object is not allowed to achieve any processing except the processing defined by its components. Emergent properties are represented as properties of the dummy component for the same reason. This further justifies the introduction of the dummy component printer core.

As one may see with this example, only the structure and the inherent properties of a composite object can be nicely represented in Darwin. This is done by using component inclusion and bindings. Aggregate and emergent properties require the introduction of a dummy component as well as a judicious usage of bindings. However, in this representation, the concept of binding is key for representing the structure, inherent, aggregate and emergent properties of a composite object. The dummy component which is introduced to handle aggregate and emergent properties is similar to the concept of a dominant object in OMT. According to Rumbaugh (Rumbaugh, 1994), the dominant object in a composite object is a component which holds information common to the entire composite object.

5 DISCUSSION AND CONCLUSIONS

The lesson of this experience is that the concepts of (1) structure and (2) inherent and aggregate properties of a composite object can be represented using the same artifact. In the context of RM-ODP, we use the concept of contract and in Darwin we use the concept of binding. These two concepts represent (structural or behavioral) interconnections between objects. In fact, this observation shows how we may interpret significant aspects of composition of objects in terms of interconnections of objects.

This raises the question whether a distinction should be made between composition and interconnection of objects. A quick tour of related literature leads to the conclusion that this distinction is hard to establish (Johnson and Opdyke, 1993), (Rumbaugh, 1995) and (Cook and Daniels, 1994). To quote Cook and Daniels: "Composition of objects is a concept which is

unhelpfully vague" (Cook and Daniels, 1994). Furthermore, in (Rumbaugh, 1995), Rumbaugh states that the semantic usage of aggregation (composition) is in recursive data structures, where it can be used to forbid cycles among the instances. In general, if you are not sure when to use aggregation, Rumbaugh proposes to ignore it and use association in place of composition.

Another aspect of the confusion between composition and interconnection of objects is the question on the semantics of composition. To what extent composition of objects affects the way we build distributed systems? This is an interesting question. Exploring the issues related to this question is out of the scope of this short paper. However, we may restrict the question to "Is the concept of hierarchical composition analogous to that of dynamic interactions between objects?" We believe that composition of objects embeds hierarchical composition as well as dynamic interactions. This is reflected in the definition of a composite object as consisting of a structure in terms of components and interactions among these components. These interactions are also reflected in aggregate properties of the composite object. As a consequence, we suggest design practices where hierarchical composition is distinct from dynamic interactions. Hierarchical composition implies the abstraction of the set of components. These components form a new object which may have properties which are different from those of its components (e.g. aggregate and emergent properties). In addition, in the context of hierarchical composition, we may make some of the components visible and hide the others while in an interconnection of objects all the objects remain visible.

In fact, it is up to the specifier to determine which aspect of composition will prevail in the modeling of an application or if both are required. Next to this, he has to make the appropriate interpretations in terms of the specification framework. It is also noted that one may view object composition as the combination of interconnection of objects and abstraction. In general, when we want to establish a linkage between properties which are defined at adjacent levels of abstraction, we need an abstraction function (Liskov and Wing, 1993). The abstraction function allows the translation of properties of a given level of abstraction into properties of another level of abstraction. This is the role played by the concepts of inherent and aggregate properties. These concepts relate composite object properties to component properties. The appropriateness of abstraction, i.e. viewing the component objects as a unit (the composite) determines whether we may consider a set of interconnected objects as a composite object.

In RM-ODP and Darwin the linkage between the structure and the behavior of composite objects is implicit and incomplete. These frameworks do not emphasize this linkage which is an important aspect of object composition. *Composites have no additional semantics but are merely specification artifacts which do not necessarily exist in the application being modeled.*

In this paper, we have shown the analogy between composite objects and distributed systems. Grounded on this analogy, we use the concept of a composite object as the starting point for assessing the accuracy and the adequacy of two frameworks for the description of distributed systems, namely RM-ODP and Darwin. *The experience with the interpretation of composite objects using these frameworks teaches us that there is still a confusion between composition and interconnection of objects. This is caused by the fact that the semantics of object composition includes the concept of interconnection of objects.* Finally, we propose that these frameworks have to be enhanced to explicitly take advantage of the linkage between structure and behavior of composite objects. This can be achieved by including the concepts of inherent, aggregate and emergent properties within these frameworks and considering that an object may have an internal structure in terms of components and interactions among these components.

6 REFERENCES

Booch, G. (1994) *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Co. Inc.

Cargill, T. (1991) A Case Against Multiple Inheritance in C++. Proceedings of USENIX Conference.

Coleman et al. (1994) *Object-oriented Development, THE FUSION METHOD.* Prentice-Hall.

ISO-1 (1995) Information technology -- Basic reference model of Open Distributed Processing, Part 1: Overview, ISO/IEC DIS 10746-1, International Organization for Standardization.

ISO-2 (1995) Information technology -- Basic reference model of Open Distributed Processing, Part 2: Foundations, ISO/IEC DIS 10746-2, International Organization for Standardization.

ISO-3 (1995) Information technology -- Basic reference model of Open Distributed Processing, Part 3: Architecture, ISO/IEC DIS 10746-3, International Organization for Standardization.

ISO-4 (1995) Information technology -- Basic reference model of Open Distributed Processing, Part 4: Architectural Semantics, ISO/IEC DIS 10746-4, International Organization for Standardization.

Johnson, R. and Opdyke, W. (1993) Refactoring and Aggregation. Proceedings of Object Technologies for Advanced Software, Nishio, S. and Yonezawa, A. (Eds.) LNCS 742, 264-78.

Kramer, J., Magee, J., Sloman, M. and Dulay, N. (1992) Configuring object-based distributed programs in REX. Software Engineering Journal, 139-49.

Liskov, B. and Wing, J. (1993) A new definition of subtyping. OOPSLA'93.

Ramazani, D. and Bochmann, G.v. (1995) A Conceptual Framework For Object Composition and Dynamic Behavior Description. Publication départementale #949, DIRO, Université de Montréal, Montréal, Canada.

Robinson, P. (1992) *Hierarchical Object-oriented Design.* Chapman & Hall.

Rumbaugh, J. et al. (1991) *Object-oriented Modeling and design.* Prentice Hall.

Rumbaugh, J. (1993) Disinherited! Examples of misuse of inheritance. *JOOP* Vol. 5, No. 9, 22-4.

Rumbaugh, J. (1994) Building Boxes: Composite Objects. *JOOP* Vol. 7, No. 7, 12-22.

Rumbaugh, J. (1995) OMT: The object model. *JOOP* Vol. 7, No. 8, 21-7.

Sakkinen, M. (1989) Disciplined Inheritance. Proceedings of ECOOP Conference, 39-56.

Cook, S. and Daniels, J. (1994) Designing Object Systems: Software isn't the real world. *JOOP* May, 22-8.