How can a subsystem be both a package and a classifier?

Joaquin Miller [1] and Rebecca Wirfs-Brock [2]

[1] EDS, 10 South 5th Street Suite 1100
Minneapolis MN 55402 USA
joaquin@acm.org

[2] Wirfs-Brock Associates, 24003 SW Baker Road
Sherwood OR 97140 USA
rebecca@wirfs-brock.com

Abstract. The UML specifies that a subsystem is both a package and a classifier. This paper explores what that could possibly mean and explains why that was the right choice. It points out a key to the use of the concept in CASE tools, mentions the historical precedent for that key, and challenges CASE tools to support the flexibility that architects and designers need. Along the way, the paper reviews a method for discovering a good partition of a system into subsystems, describes a scheme for using UML to build a model of a system, and suggests some changes to the UML.

System has two clusters of meaning in English:

I: An organized or connected group of objects.
II: A set of principles, a scheme, method. [1]

We describe a system for modeling systems. That is: We review a method for discovering a good partition of a system into subsystems. We offer a set of principles for use of the concept, system, in specifying an organized or connected group of objects. And we describe a scheme for building a specification of a system. Then we pose some challenges for CASE tool builders.

We will use ‘subsystem,’ as shorthand for ‘instance of a UML Subsystem.’

The purpose of this contribution is to clarify (one interpretation of) the meaning of ‘subsystem’, and to challenge tool builders to provide excellent support for using subsystems in models.

The first section briefly states what we intend by ‘subsystem.’ This will narrow the subject of this paper to the meaning we have in mind.

The second section reviews one method for using subsystems to design a system. This method discussion, though not the point of this paper, is relevant, because it is the process of using subsystems in design that creates the need for the tool support we challenge tool builders to offer.

In the third section we assert some principles.

The body of the paper, in the fourth section: explains our interpretation of the UML concept, subsystem; answers the question in the title of this paper; takes the reader through some model changes that might take place using subsystems to design; and makes a brief comment on UML notation.

Finally, we present our challenges.

1 What is a subsystem?

A system is an organized or connected set of parts. A subsystem is a kind of system. Because it is a system, a subsystem has parts. It is a subsystem because it happens to be a part of a larger system.

The concept, subsystem, serves as a tool for both organization and abstraction. Used for organization, a subsystem is a tool for partitioning a system: a subsystem represents one of the parts of the system. Used for abstraction, a subsystem is a tool for hiding complexity: a subsystem hides its own parts, that is, the details of its internal structure.

The UML Subsystem represents a combination of and a compromise between different needs felt by several of the UML partners.[1] We discuss only the needs for organization and abstraction in designing and presenting a specification. We use ‘subsystem’ only in the sense described in this section. (We will briefly discuss other uses of UML Subsystem near the end.)

2 Review of a method

Partitioning a system is one of the ways of reducing complexity by separating concerns. Dividing a system into subsystems not only reduces complexity for architects and designers and simplifies the work of programmers; it also provides project managers a way to organize the work of development.

Principles for discovering a good partition have been well known for decades. Myers identified the central concepts as module, coupling, and strength (now called cohesion). [3] These concepts have since been restated in terms of objects. [4], [5] Many other authors have made contributions to understanding subsystems.

For two reasons we briefly review a method for partitioning a system. This will provide a background for the principles presented in Section 3. It will also motivate the need for the tool support we challenge tool builders to offer.

We choose a particular method, [5], but another would serve as well for the purposes of this paper.

1 Hewlett-Packard, IBM, ICON Computing, i-Logix, IntelliCorp, EDS, ObjecTime, Oracle, Platinum Technology, Ptech, Rational Software, Reich Technologies, Softeam, Sterling Software, Taskon and Unisys.

2.1 Things to do

Divide responsibilities. Identify a set of subsystems and assign responsibilities to each of them. Evenly distribute system intelligence. Specify responsibilities as generally as possible. Keep behavior with related information. Keep information about one thing in one place. Share responsibilities among related objects.

Model Interactions. In the context of the collaborating subsystems, identify all the interactions between subsystems.

Simplify. Minimize the number of interactions a subsystem has with other subsystems. Minimize the number of other subsystems to which a subsystem delegates. Minimize the number of interfaces that a subsystem presents.

Evaluate the result. The goal is to maximize the cohesion of each subsystem and minimize the coupling between subsystems. Ask if there is opportunity to improve the design.

Repeat. Repeat the process, making changes to the design at each step. Continue design iterations while significant improvements are found.

Of course, it is good to build and execute architecture prototypes as a part of thi process; this is very likely to uncover reasons to change the design. And, as parts of the design appear to be stable, development may start. Certainly, this process will, in any case, continue while development is underway, as the surprises come.

2.2 Direction of work

The previous section discusses the work of designing subsystems. But that work can start from different places and proceed in different directions.

Bottom up. In [5], designers were advised to find classes, then to build and refactor class hierarchies, and finally to identify subsystems. This approach gives the subsystem designer a well thought out and well specified set of parts to use in specifying subsystems.

Top down. Critics of [5] said that subsystems should be identified first before any objects or classes are modeled. Architecture should precede design at the object level, they said, or the design method will not scale to large systems. Many prefer an approach that starts with the responsibilities of the system, then designs a set of subsystems, and specifies objects (and classes) only later in the process.

Middle out. In practice, a lot of work is actually done starting with a small number of potential or candidate subsystems and working in both directions. To better understand or further specify our model we may partition one of the parts of a subsystem, turning an object into a subsystem. And to gain a more abstract view, we may pick several subsystems and make them the parts of a larger subsystem. Or we may find a generalization that permits one subsystem to do the work that was done by two. Furthermore, subsystems or objects that are part of one subsystem may be moved to another.

This is the most practical approach for many problems.

3 Principles

The argument of the paper is based on certain principles, which the authors hold and present here simply as assertions.

A system is “something of interest as a whole or as comprised of parts. … A component of a system may itself be a system, in which case it may be called a subsystem.” [2]

“The basic elements of architectural description are components,[2] connectors, and configurations.” [6] In object modeling, a component of a system is a subsystem or an object. (A connector, if it is of interest as consisting of parts, is a subsystem, too!)

During the process of designing a system, many changes will be made:
— Any object may become a subsystem as the design develops.
— Likewise, any subsystem may become an object. That is, the parts of the subsystem may be combined into a single object.
— We may replace a subsystem with some already specified object, or with a complex part that we treat as a simple object in our design.
— We may add an object to a model, but not know whether it will later become a subsystem. (That is, not know whether it will later come to be of interest as composed of parts.)
— After making an object or subsystem a part of one subsystem, we may move it to another subsystem.

Often, we will start to do something, but not be ready to complete it:
— We may fasten a connector to an object, but not know where we will connect the other end.
— Or feel an urge to specify a connector without connecting it to anything yet.
— We may add objects to models, without (yet) specifying that they are a part of a particular subsystem.

Always, we will use the power of abstraction. In particular:
— At a given level of abstraction, we may wish to hide the fact that something is a subsystem. (That is, hide the fact that it has parts.)

All these changes will be intermingled with shifts in viewpoint or level of abstraction. Neither our modeling language nor or methods, nor our tools have any reason for limiting these changes in any way. Rather, they must enable them, and make them easy for us.

2 ‘Component’ in the usual English sense: a constituent element or part. The authors of [2] and [6] do not mean UML components.

4 A Scheme

In the UML, a subsystem “represents a behavioral unit.”[3] It is defined to be both a classifier and a package. The UML classifier is an abstraction used to specify the features common to classes, interfaces and other UML model elements. “A classifier … describes behavioral and structural features.” “The purpose of the [UML package] is to provide a general grouping mechanism.” “In fact, its only meaning is to define a namespace for its contents.”

At first glance, it is a bit of a puzzle that something could be only a way to group things and define a namespace for them, and at the same time be a way to describe behavior. But we will show that the UML partners made the right choice.

The model elements grouped by a UML Subsystem into three categories:
— Operations
— Specification element
— Realization elements

We will mention each of these in the next section.

4.1 Levels of description

Sometimes, we want to represent a subsystem as a single object, hiding the parts; at other times, we want to show the parts.

Outside. From the outside, a subsystem is treated as a single model element. It appears as a whole, collaborating with other parts of the system to fulfill its responsibilities. Its collaborators treat the subsystem as a black box. Thought of in this way, subsystems are yet another encapsulation mechanism. The services provided by a subsystem are represented by interfaces and the corresponding operations. Other model elements further specify the behavior. (For example, a state machine, perhaps with action specifications.) These other model elements are what UML calls the specification elements of the subsystem. The operations and the specification elements of the subsystem (as well as any interfaces defined for the subsystem) are what specify the system without reference to its parts.

An instance of a UML classifier is an appropriate representation of a subsyste when it is being treated as a whole and as a black box.

Inside. From the inside, a subsystem reveals itself to have a complex structure. It is a system of objects collaborating with each other to fulfill distinct responsibilities that contribute to the purpose of the subsystem: the fulfillment of its responsibilities. These model elements specify the subsystem in terms of its parts; they are what UML calls the realization elements of the subsystem.

---

3 Quotations for which no reference is given are from the OMG UML specification current at this writing, [7]. The latest specification is available at: http://uml.shl.com .

A UML package is an appropriate container for the objects, relationships and other model elements that are the parts of a subsystem. The package will also hold other model elements that help specify the way these parts work together, for example, collaborations and interactions.

Outside	Inside
Interfaces Operations Specification elements	Realization elements

4.2 Solution to the puzzle

Abstraction is “the process of suppressing irrelevant detail to establish a simplified model.” [2]

We see the escape from our puzzle: how can anything be both a package and a classifier? For us, as architects or designers, a UML Subsystem is***not*** both a package and a classifier. A UML Subsystem is ***either*** a package or a classifier, depending on our level of abstraction.

[As mentioned above, we use ‘subsystem,’ as shorthand for ‘instance of a UML Subsystem.’ So we rephrase: A subsystem (an instance of a UML Subsystem) is ***either*** a package or an object, depending on our level of abstraction.]

The relationship of the subsystem considered as a single model element to the objects and relationships that are the parts of that subsystem is that of abstraction. The subsystem is an abstraction of its parts.

We can think of a subsystem as simply an object, if we view it from a particular level of abstraction and suppress the details of its realization. Those details are irrelevant in the black box view. Or we can think of a subsystem as only a package, if we view it from a more detailed level, but suppress the detail that it is considered an object in another view

An object (an instance of a UML class) is an appropriate representation of a subsystem when it is being treated as a black box.

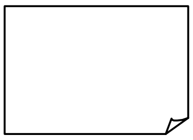
Of course, we also need to specify the correspondences between the specification of the subsystem and the specification of its parts and their collaborations.

4.3 Drawing the pictures

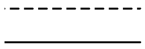
This section presents what amounts to a simple modeling language. It will be clear that what we propose can be expressed using the variety of relationship kinds, keywords and stereotypes available in UML.

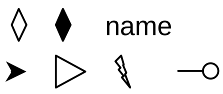
In the spirit of abstraction, we use a very simplified model. Because of our choice of words, this will appear to be a model about pictures, but actually it is a simplified model of the UML. (The pictures are not UML notation, nor a proposed alternative.)

Our model includes:


Pages, on which we draw

Boxes, representing objects, including subsystems.

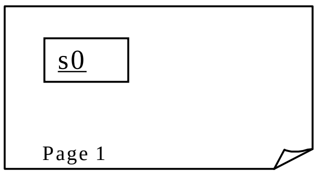

Lines, representing various kinds of connections.



Adornments, which can mean many things.

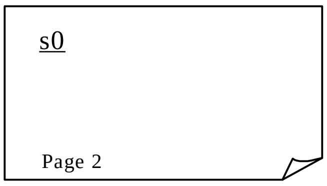
In this simplified model, building a specification is represented as drawing boxes on pages, connecting the boxes with lines, and placing adornments on the lines. [The drawings of boxes, lines, and adornments in this simplified model are not diagrams in UML notation. But the reader (in particular, CASE tool makers) will see what we are getting at.]

Let us now illustrate how we might move between levels of abstraction and view of our model as we build one such specification. To start, we draw a box on a page and give it a name. This box represents an object, s0. At some point, we decide that s0 is a subsystem. It still appears as a box, because we are viewing it from outside, as a black box.



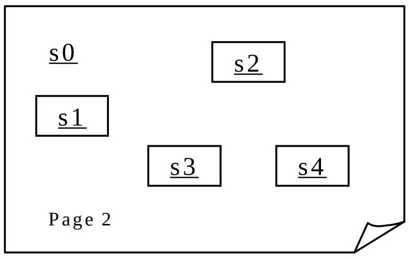
A box on a page.

When we wish to specify the parts of s0, we zoom in, and s0, which was a box in the view above, and becomes a page, as shown below.



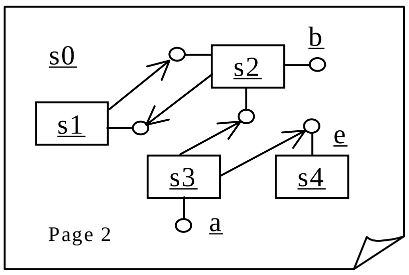
The box becomes a page.

We will use this page to specify s0 when seen from the viewpoint that considers it as being composed of parts. Since no parts yet exist, we specify them and add them to our model.



We specify the parts of s0.

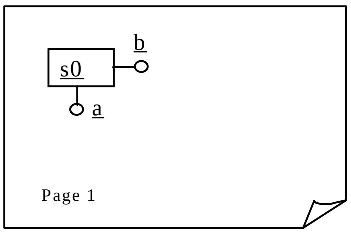
If we zoom back out, we will see s0 as a box, unchanged. If we decide that s3 is a subsystem, we can zoom in on it, and we will see another page, labeled s3, where we can specify the parts of s3. But let's zoom back out and specify the connections of the parts of s0.



We connect the parts.

[In this diagram, the lines are connectors, and mean: is connected to. The UML provides a rich selection of relationships and relationship stereotypes.]

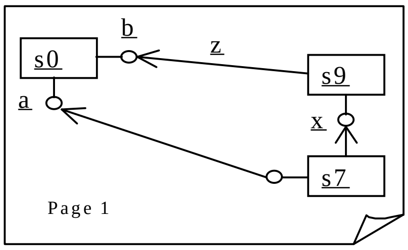
When we zoom back out again, what do we want to see? Not the parts of s0, and not the connections of the parts. But we do want to be able to specify that the two unconnected interfaces, a and b, are to appear as interfaces of s0.



We zoom back out.

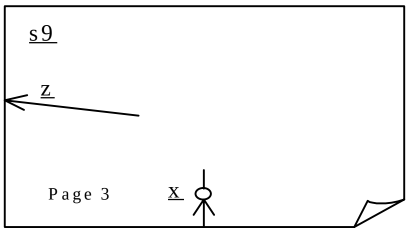
[Reminder: These are drawings in our simplified model, not diagrams using UML notation.]

Now we will work on our design at this level of abstraction. We add some objects to this page. We specify interfaces on the new objects, and make some connections from the objects to subsystem s0.



We add some more parts and connections.

Now we decide to make s9 a subsystem. [It means we decide that we are interested in s9 as composed of parts.] We zoom in on s9. We see those connectors that are connected to s9, and we know that connections must be made to some part or parts of subsystem s9.



We zoom in on a different part.

What we want to see on when we zoom in on a box that has lines connected to it are the lines, “adornments attached to the main part of the path” and “adornments at [the] end where the path connects to the” box.

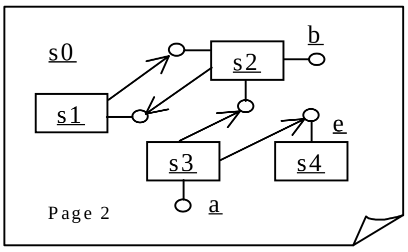
Obviously, this requires either:

— relaxing the current UML requirement that “paths are always attached to other graphic symbols at both ends (no dangling lines),” or

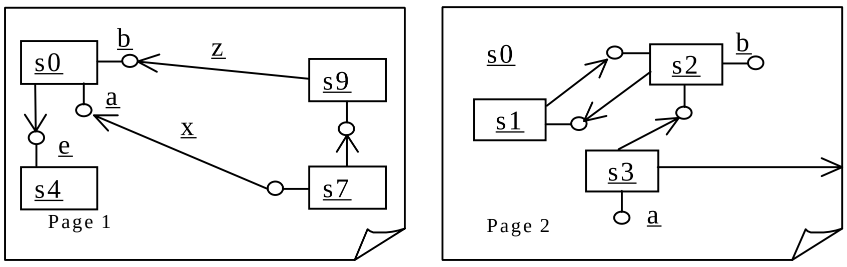
— adding a new graphic symbol to represent the model element at the other end of the line (perhaps a small open pentagon, which is widely used as a off page connector symbol).

Adding a new graphic symbol would provide a place to attach a hyperlink. Following that hyperlink would lead to another page, the outer diagram, focused on the model element at the other end of the line.

We also need to be able to easily change our mind about our design. Consider the parts of the subsystem, s0, again.



We must be free to decide that we want s4 to be, not a part of s0, but another subsystem appearing as a peer of s0. When s4 is moved, the model will be:



This should be a single step for the modeler, with the modeling tool making the several necessary changes.

As we said at the start of this section, this model is about what we want to specify with UML. Obviously, we would like our drawing tool to work this way also.

4.4 UML Notation

We believe it will be a clarifying improvement to consider a subsystem to be a kind of object (or, when reduced to code, an instance of a component) rather than an instance of some other kind of classifier. (When not reduced to code, the subsystem is an abstraction, since some details are hidden, since it is a simplified model.)

UML does not consider a subsystem to be an object. However, the ideas above can be implemented while adhering to the adopted UML specification.

The three-compartment notation allows operations, specification elements and realization elements to be shown at once in separate compartments. But any of the compartments can be suppressed.

4.5 Desirable subsystem properties

A modelling language needs to provide rich capabilities for specifying subsystems. UML already deals with some aspects quite nicely, others it left out.

Attributes. UML specifies that a subsystem may not have attributes. But it is often very useful to attribute attributes to a subsystem. This should present no problem, especially if we are able to see an attribute as an abstraction, rather than as a data member of a programming language object. We mean to say that an attribute can and should be expressed at the same level as its object. So, if we think of a subsystem as an abstraction, then attributes should not be understood as code-specific declarations. Instead, for example, as responsibilities of the subsystem.

The specifier of such an abstract attribute will then need to show how it is related to the realization elements of the subsystem. (For example, a designer might specify that the value of an abstract attribute is derivable from the values of some attributes of some objects among the realization elements. Or an attribute of a subsystem might be refined as an operation on an object among the realization elements.)

We can use such an abstract attribute, for example, in the specification of an invariant or a state machine; a tool simulating a model could display the values of such attributes. Yet another reason it will be useful to model a subsystem as an object.

Interfaces needed. In addition to specifying the interfaces a subsystem provides, we need to specify those it needs. This may be specified in UML. A more symmetrical approach would treat provided and needed interfaces uniformly. [8] and [9] show how to do this.

Component. When construction of a system reaches the point where we have code, UML adds another concept, component. A subsystem in the specification may appear as a component in the code. The parts of a subsystem may appear as components in the code.[4] If a subsystem is implemented as a component, it is proper and correct to call this component a subsystem.

Generalization. A UML Subsystem is a generalizable element. That's good. Being a generalizable element, a subsystem may be abstract or not. That's good, too.

Instantiability. A UML Subsystem may be instantiable or not. This is not the place to discuss what the UML specification might mean by 'is instantiable.' Nor whether there might be a difference between having an instance in a model, and having an instance in the universe of discourse of the model (for example, at run time). (Whether there might be an instance in a model that did not correspond to a concrete thing in the universe of discourse. Whether a model might include an instance of an abstract class.)

However all that may be, allow us to mention some possibilities:

In an implementation using the Façade pattern, a subsystem will appear at run time as a single programming language object that corresponds to the outside view of the subsystem. (This façade object will be a realization element of the subsystem.)

Following CORBA, a subsystem might appear at run time as a CORBA object.

If a component platform is used, a subsystem might appear at run time as an Enterprise Java Bean or CORBA component.

If whatever implements a subsystem at run time maintains an IP address and port, or corresponds to a CORBA object reference, it has a unique identifier, even if there is no single object that is the subsystem.

In these cases, in many other cases, and in all the cases that fall under our use of the concept, 'subsystem,' a subsystem at run time will have the property that distinguishes it from all other things: the quality that it is itself, and not something else. It will have identity. [1], [10]

That does not mean that at run time a subsystem will always comprise a single object or component or even that it will have an identifier. It will often be the case that only the objects from the inside of the subsystem (the objects that are realization ele-

4 Unless these two possibilities are prohibited by the UML well-formedness rule that: "A Component may only implement DataTypes, Interfaces, Classes, Associations, Dependencies, Constraints, Signals, DataValues and Objects."

ments of the subsystem) will appear as objects at run time. In that case, the run time subsystem comprises this collection of objects and their links.

Abstraction. We ask the reader to agree with us that abstraction is useful even when talking about run time, about “physical,” about the real world. Whether there is an object at run time that corresponds to (is an instance of) a given subsystem in a model is both a question of levels of abstraction and an implementation decision. Someone might ask: “Is there is an object at run time that corresponds to this subsystem in the model?” As to levels of abstraction, a quite reasonable answer is: “That depends on what you mean by object.” As to implementation decisions, a quite reasonable answer is: “We haven’t decided yet.”

There is no reason for UML to dictate the set of possibilities. The builders of each specification that uses UML will say what they mean.

4.6 Just what is a subsystem, anyway

We have presented our understanding of (one use of) the UML concept, subsystem. It is not easy to tell from the UML specification if this is included in what it wants to mean by subsystem. The structure (“abstract syntax”) is clear. And the discussion of the specification and realization elements is as we have described. But the meaning (“semantics”) of the concept is murky.

How it happened. To testify to history:[5] Among the UML partners different needs were felt, all of which needs could be met by something called ‘subsystem.’ Some felt a need for “a grouping mechanism for specifying a behavioral unit of a physical system.” Others felt a need for what we have presented as our meaning of ‘subsystem.’ There is adequate historical precedent for these different uses of the term, ‘subsystem.’ And these different uses have much in common. The UML Subsystem represents a combination of and a compromise between ideas to meet these different felt needs.

What resulted. We feel that this combination and compromise has resulted in a portmanteau concept and unclear text in the specification. For example: A subsystem has operations, but “... has no behavior of its own.” A subsystem may be instantiable, but “... there are no explicit instances of a subsystem; instead, the instances of the model elements within the subsystem form an implicit composition to an implicit subsystem instance, whether or not it is actually implemented.”

It appears to us that this language wants to describe the subsystem when used “merely as a specification unit for the behavior of its contained model elements.”

What to do. We suggest the UML define ‘subsystem’ along the lines of what RM-ODP calls a composite object: an object expressed as “a combination of two or more objects yielding a new object, at a different level of abstraction.” [2] From the outside

5 Personal experience of one of the authors (Miller).

viewpoint a subsystem will be an object. From the inside viewpoint it will be a set objects and links. (Those who prefer the style will use UML classes and associations.)

When the object seen from the outside viewpoint is intended “merely as a specification unit for the behavior of” the objects seen from the inside viewpoint, and not to be thought of as existing at run time, the model will specify that. A more down home concept than is not instantiable would serve that purpose.

When we decide an object is a subsystem (has parts), let the tool create a package for the parts.When we decide we want to compose a subsystem from a set of objects we have already specified, let the tool put them in a package we can view as an object.

5 Challenges

Architects and designers need CASE tools that handle the UML Subsystem in a way which reinforces the principles we have offered and which provide strong support for designing subsystems in a middle-out, top-down or bottom-up fashion.

As illustrated by the review of a method in Section 2, the typical process of designing subsystems is iterative and involves restructuring the design. During the design process, objects (or the classes they are instances of) will be moved from one subsystem to another; subsystems will be removed from the design and others added; what was a subsystem will become an simple object, and what was a simple object will become a subsystem.

We challenge the builders of CASE tools to make the restructuring of a design a easy as possible. Build your tools so it is easy for architects and designers to:

— Shift levels of abstraction (with connections mapped and preserved by the tool).
— Move objects and subsystems from one subsystem to another (as easily as cut and paste or drag and drop).
— Change an object into a subsystem (as easily as selecting a menu item).
— Change a subsystem into an object (and have something helpful done with what were the parts of the subsystem).
— Combine a set of objects into a subsystem.

For the day that those challenges are met, here are some easy ones. Architects and designers will be well served by a UML and CASE tools that allow them to:

— Treat a subsystem as a full-fledged object, when viewed from outside.
— Specify attributes for a subsystem, to represent what [5] calls responsibilities for knowing, to be used in specifying invariants or state machines, or to be displayed by model simulators.
— Specify the interfaces that a subsystem requires, using a graphical notation as simple and compact as the lollipop, which indicates that an interface is provided.
— Having specified that a subsystem requires a particular interface, connect the graphical representation of that requirement to the graphical representation of an interface on another subsystem that satisfies the requirement.

6 Conclusions

Architects and designers think at multiple levels of abstraction. This means they see subsystems both as objects and as containers of parts. They find it natural to shift focus between these two viewpoints. The ideas we have presented in this paper are not new. Years ago, when functional decomposition was popular, data flow diagrams were used. In those old, slow days, this zooming in and out, with connections preserved and mapped, is exactly how good data flow modeling tools worked.

With modest extensions to UML, and with serious effort by toolmakers, UML and CASE tools can better support working at multiple levels of abstraction.

7 Acknowledgements

Thanks to all the reviewers for their comments. Special thanks to Jim Rumbaugh for his extensive criticisms, helpful suggestions and encouragement. And many thanks to all the authors whose ideas have contributed to our thinking.

References

1. Oxford English Dictionary (Second Edition), Oxford University Press, Oxford (1994)
2. Information Processing–Open Distributed Processing–Reference Model–Foundations, X.902 | IS 10746-2. International Organization for Standardization, Geneva (1995) http://enterprise.shl.com/RM-ODP/
3. Myers, G. J.: Reliable Software through Composite Design. Van Nostrand Reinhold, New York (1975)
4. Cox, B. J., Novobilski, A. J.: Object-oriented Programming–An Evolutionary Approach. Addison-Wesley, Menlo Park, California (1986)
5. Wirfs-Brock, R., Wilkerson, B., Wiener, L.: Designing Object-Oriented Software. Prentice Hall, Englewood Cliffs (1990)
6. Shaw, M., Garlan, D.: Software Architecture–Perspectives on an Emerging Discipline. Prentice-Hall, Englewood Cliffs (1996)
7. Object Management Group, OMG Unified Modeling Language Specification—Version 1.3. Object Management Group, Framingham, Massachusetts (1999) http://www.omg.org/cgi-bin/doc?ad/99-06-08
8. Reenskaug, W., Reenskaug, T., Lehne, O. A.: Working With Objects: The OOram Software Engineering Method. Prentice Hall, Englewood Cliffs (1995)
9. Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modeling. John Wiley & Sons (1994)
10. Khoshafian, S. N., Copeland, G.P.: Object Identity, OOPSLA '86 Proceedings, in SIGPLAN Notices 21, 11 (1986) 406-416.