

# Semantics of Navigability in UML Associations

Gonzalo Génova

Computer Science Department, Carlos III University of Madrid,  
Avda. Universidad 30, 28911 Leganés (Madrid), Spain

E-mail: ggenova@inf.uc3m.es

<http://www.inf.uc3m.es/>

**Resumen.** El concepto de navegabilidad de asociaciones en UML está pobremente explicado en la documentación oficial. Este artículo intenta expresar clara y concisamente la asimetría conceptual de las asociaciones que es expresada mediante la navegabilidad, examinando su relación con otros elementos del lenguaje (dirección de nombre de asociación, visibilidad), mostrando las principales consecuencias de que una asociación sea navegable (capacidad de enviar mensajes, dependencia inducida), explorando las propiedades que dependen de la navegabilidad (invertibilidad de la asociación, eficiencia de navegación), y definiendo algunas preferencias sobre la notación de la navegabilidad. Este análisis de la navegabilidad puede ayudar a resolver el dilema "asociaciones bidireccionales frente a asociaciones unidireccionales", al revelar algunos aspectos centrales del concepto de asociación en tanto que "conocimiento".

**Abstract.** The concept of navigability of associations in UML is poorly explained in the official documentation. This paper tries to express clearly and concisely the conceptual asymmetry of associations that is expressed as navigability, examining its relation to other language elements (association name direction and visibility), showing the main consequences of an association being navigable (ability to send messages and induced dependency), exploring properties that depend on navigability (association invertibility and navigation efficiency), and defining some preferences about the notation of navigability. This analysis of navigability can help to solve the "bidirectional versus unidirectional associations" dilemma, by revealing some central aspects of the concept of association as "knowledge".

## 1 Introduction

During the last decade there has been a intense controversy since James Rumbaugh introduced in object-oriented models a strong concept of association derived from entity-relationship models. In this approach, associations should be regarded as first-class semantic constructs of similar weight to classes and generalization, because classes and associations abstract jointly, and in a natural way, not only the high-level static structure of the system, but also the overall structure of interactions between objects. Consequently, object-oriented languages should evolve to implement better this construct. The original Object-Relation model [Rumbaugh, 1987] derived into the Object Modeling Technique [Rumbaugh et al., 1991], which finally was one of the main conceptual and notational sources for the Unified Modeling Language [Object Management Group, 2001]. This view has its stronger opponents in researchers such as Brian Henderson-Sellers and other promoters of the OPEN methodology [Graham et al., 1997b]. They criticize UML as excessively based on data modeling, with a bidirectional concept of

association that violates the principle of encapsulation, and thus compromises reuse [Henderson-Sellers and Firesmith, 1999].

In the course of this controversy, the logical and implementation levels have been too often erroneously mixed, especially when the debate has been focused on a supposed opposition between the use of associations and the use of attributes in modeling, as though we could choose to use associations and discard attributes, or vice versa [Graham et al., 1997a; Tanzer, 1995; Velho, 1994]. Both associations and attributes are logical constructs that are necessary for developing a good analysis and design, and their semantic difference is found at the logical level, not at the implementation level. This difference is best understood from the concept of identity, such as Rumbaugh puts it: "use associations to show interobject references that have identity, and use attributes to show encapsulated values that have no identity" [Rumbaugh, 1996b]. There is a loose correspondence between the couple association/attribute at the logical level, and the couple tuple/pointer at the implementation level, but they remain nevertheless independent: you can implement an association either with tuples or with pointers, and the same is valid for an attribute.

Associations are necessary in OO modeling. Without them we would not have encapsulated objects, but isolated objects that cannot interact. Every association in a model breaks somehow the principle of encapsulation, since it introduces a dependency between the associated objects that cannot be avoided if we want them to work together. A bidirectional association induces a mutual dependency that is worse than the one-way dependency of a unidirectional association, but, in spite of this, UML favors bidirectionality as a default option. In this sense the criticism from the side of the OPEN Consortium is reasonable and should not be ignored: it could serve to avoid abuse of bidirectional associations, and thus it could result in more reusable models.

In our opinion, then, the crucial question is not "associations versus attributes", but "bidirectional versus unidirectional associations". In this spirit, we have tried to clarify the issue of directionality of associations, which is the subject of this paper. Our analysis will remain at the logical level, in a manner which is basically independent of implementation details, that is, of the concrete way of implementing associations. The UML concept that more directly expresses the directionality of an association is "navigability", which is graphically expressed as an open arrow at the end of the association line that connects two classes, pointing to the direction of traversal. Navigability is closely related to the ability of sending messages, so that very often this two concepts are identified. In fact, navigability is not immediately the direction in which messages can fly, but the direction in which the sender can look out.

The remainder of this paper is organized as follows. Section 2 examines two other UML concepts that are in close proximity to navigability, namely association name direction and visibility; before telling what navigability is, we will tell what it is not. Section 3 searches an "authorized" definition of navigability in the official documentation, which leads to the concept of "navigation expression". Section 4 proceeds by showing the two main consequences of navigability, which are the ability of sending messages through navigable associations, and the dependency between classes induced by navigable associations. Section 5 explores two properties of navigable associations, invertibility and efficiency, the study of which can serve to elucidate the essence of the concept of association, which is the ultimate aim of our research. Finally, section 6 reasons about our preferences in using the UML notation of navigability.

Since this is a conceptual research about the official definition of navigability, our main sources have been The Unified Modeling Language User Guide [Booch et al., 1998], The Unified Modeling Language Reference Manual [Rumbaugh et al., 1998], and The OMG Unified Modeling Language Specification [Object Management Group, 2001], more briefly referred to

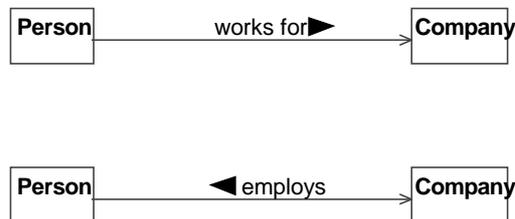
as "The Standard".<sup>1</sup> This latter document is properly the only one which is truly "official", but there are many semantic questions that are poorly explained in the Standard, and obviously the Reference Manual seems the best choice for searching a clarification. On the other side, we cite the User Guide not because we consider it a particularly reliable source, but because it is probably the main source for many modelers, so that we think it is important to show its virtues and deficiencies. We cite version 1.4 of the Standard, which is at present at a draft stage; in any case, we have checked that there have been no significant changes from version 1.3 regarding these topics.

## 2 Some related concepts

### 2.1 Name direction

The User Guide warns us that we should not confuse *association name direction* with *association navigation* [UG 66], although it does not clarify the question any more, and we must turn to the Reference Manual of the Standard for further information. The confusion is possible, since both characteristics mean some kind of directionality of the association, and both use an arrow to indicate it.

An association can have a name which is just used to describe the nature of the relationship. So that there is no ambiguity about its meaning, we can use a direction triangle that points in the direction we intend to read the name, that is, towards the class designated by the verbal construct (see Figure 1).



**Fig. 1.** Two reciprocal associations with the same navigability but with inverted name directions: a) a person works for a company; b) a company employs a person. Both associations may be regarded as two different associations, or else as two reciprocal versions of the same association

The use of a name direction triangle is related with the conceptual or *linguistic asymmetry* of the association. This is more clearly explained in the Reference Manual: "The different ends of an association are distinguishable, even if two of them involve the same class. (...) Because the ends are distinguishable, an association is not symmetric (except in special cases); the ends cannot be interchanged. This is only common sense in ordinary discourse; the subject and the object of a verb are not interchangeable" [RM 50]. We use the name direction triangle to mark this asymmetry, so that the example in Figure 1a can be read as "a person works for a company", but not as "a company works for a person". Asymmetry does not mean, in this case,

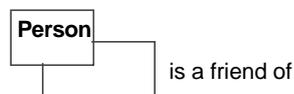
---

<sup>1</sup> In the remaining of this paper, these three references will be cited as "UG" for the User Guide [Booch et al., 1998], "RM" for the UML Reference Manual [Rumbaugh et al., 1998] and "UML" for the UML Specification [Object Management Group, 1999].

that one of the roles is more important than the other, because it is a whole made out of parts (an aggregation) or because it is a master that controls a slave (a one-way association); it only means that the human verbal expression has a subject and an object that are not interchangeable.

There is no need for a name direction property in the metamodel of associations: the association ends of an association are inherently ordered [UML 2-15], so that the name direction is the very ordering of the classifiers within the association. That is, the underlying property of the model which is related with the name direction is the ordering of the association ends [RM 155; UML 3-71]. In Figure 1a, the association end attached to `Person` is the first one in the association, and the association end attached to `Company` is the second one.

We can think that those special cases in which the Reference Manual says an association is symmetric arise when the association specifies some kind of equivalence among the associated objects, such as "is-equal-to", "is-friend-of", etc.; the two ends of the association, of course, should be attached to the same class for the objects to be interchangeable, as in the example in Figure 2. Nevertheless, even though the association may be conceptually symmetric, the metamodel forbids the interchangeability of the roles of an association: the roles are always distinguishable thanks to the inherent ordering of the association ends, even though the name direction is not represented. In other words, the UML metamodel prevents the true representation of symmetric associations, although the graphical presentation may induce to think the contrary. This means that "John is a friend of Alice" does not imply in UML that "Mary is a friend of John". For this reason, the Reference Manual should not say that there are special cases in which a UML association can be symmetric.



**Fig. 2.** An apparently symmetric association between one class and itself

The specification of the name direction is straightforward when the name of the association is a verbal phrase, which usually requires a subject and an object, as in the examples in Figure 1. A verbal phrase in the passive voice, such as "is employed by", is less clear: it must be remembered that the name direction represents the difference between subject and object from the linguistic point of view, so that the tail of the arrow is attached to the "grammatical" subject (in this example, the person), regardless of the "real" subject of the action that is represented (the company, real subject of the action "employ"). Many modelers prefer the use of rolenames instead of association names to avoid these problems. When the name of the association is a noun, such as "work" or "employment", it may be even impossible to decide properly the association name direction, since a noun has no grammatical subject (as a name of an action, a noun can have a real subject, but we have already stated that this is not the right rule to identify the name direction). Of course, we should avoid naming associations with nouns instead of verbs, but this is a real problem that has a bad solution in the case of an association class, whose name should be a verbal phrase and a noun at the same time [UML 3-80].

The Standard says that "the name-direction arrow has no semantics significance, it is purely descriptive" [UML 3-71], a statement that we consider rather confusing, since the human interpretation of the association is certainly some kind of "semantic significance". Probably, this must be understood in the sense that the name direction imposes no other constraints on the model than the ordering of the association ends; specifically, the name direction arrow and the navigability arrow may point in opposite directions, as in the example of Figure 1b. In other

words, the name direction has nothing to do with the navigation in an object model, and consequently, has nothing to do with the communication or interaction among objects.

This "has nothing to do" has been well established in the preceding paragraphs. Nevertheless, from the conceptual point of view both kinds of asymmetry, name direction and navigability, are closely related, since the association name denotes also the interaction that occurs between the classes through the association. In the case of a one-way association, a name direction that points in the opposite direction, as in the example of Figure 1b, can be misleading, so that the name in Figure 1a is preferable. In consequence, we can infer the following style rule that can make models more readable: "when an association is navigable in only one direction, the association name direction should point in the same direction as the association itself, or better use rolenames instead of association name".

## 2.2 Visibility

Visibility and navigability are different concepts, but it is possible to confuse them since both refer in some way to the ability of one object to see and use another object's features. Curiously, the Reference Manual is a victim of the semantic proximity of these two concepts: "A lack of navigability implies that the class opposite the rolename cannot "see" the association and therefore cannot use it to form an expression" [RM 355]: this is not properly a mistake, although the verb "see" should be used to define visibility rather than navigability. Another place: "The rolename may bear a visibility marker--an arrowhead--that indicates whether the element at the far end of the association can see the element attached to the rolename" [RM 415]: this is a true mistake, since an arrowhead is a navigability marker, not a visibility marker.

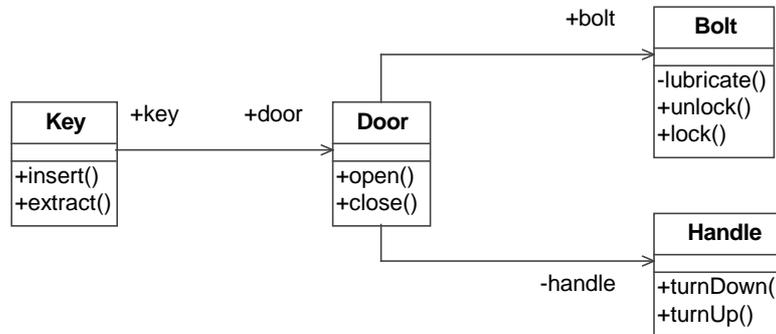
According to the User Guide, the visibility of an attribute or operation of a class specifies whether it can be used by other classes. Each feature (attribute or operation) can be marked as public (+), protected (#) or private (-), meaning that the feature can be used, respectively, by any outside class with visibility (and, we can add, navigability) to the given classifier, by any descendant of the class, or only by the class itself [UG 123]. The information given by the Reference Manual [RM 497] and the Standard [UML 2-41; 3-44] is substantially the same. Version 1.4 of the Standard adds a new kind of visibility, package (~), meaning that the feature can be used by any class in the same package<sup>2</sup>.

The visibility of an association end is defined exactly in the same way as that of a feature, but it has only sense when the association end is navigable: "If navigability is true, then the association defines a pseudoattribute of the class that is on the end opposite the rolename--that is, the rolename may be used in expressions similar to an attribute of the class to obtain values" [RM 354]. In other words, *association navigability and association visibility are independent concepts that are specified also independently, but the ability to use a certain class, feature or association end must be allowed jointly by both characteristics, primarily by the navigability and secondarily by the visibility.*

---

<sup>2</sup> But, if public visibility and association navigability are needed to use some feature, we don't see the point of having package visibility. It cannot mean "use without need of being associated", because this is already covered by public visibility through links that are not instances of associations. It cannot mean "use without need of any kind of knowledge of the target" (i.e., based on navigability), simply because this has no sense.

To illustrate the interlacement between visibility and navigability, let's consider the example in Figure 3. An object of class Door can use the operations `self.open`, `self.close`, `bolt.unlock`, `bolt.lock`, `handle.turnDown` and `handle.turnUp`; it cannot use `key.insert` nor `key.extract` because the association is not navigable towards class Key; it cannot use the operation `bolt.lubricate` due to its private visibility; it can use the operations `handle.turnDown` and `handle.turnUp`, apparently in spite of the private visibility of the association end `handle`, because the Door class is the owner of that association end, exactly as if it were his own attribute, so that a private visibility does not affect it. An object of class Bolt can use the operations `self.lubricate`, `self.lock` and `self.unlock`; it cannot use other operations due to the lack of navigability in its association with Door. Similarly, an object of class Handle can use only the operations `self.turnDown` and `self.turnUp`. Finally, an object of class Key can use the operations `self.insert`, `self.extract`, `door.open`, `door.close` and, thanks to the public visibility of role `bolt`, it can use also `door.bolt.unlock` and `door.bolt.lock`; it cannot use `door.bolt.lubricate` due to the private visibility of the operation; it cannot use the operations `door.handle.turnDown` and `door.handle.turnUp` because the role `handle` is private to objects of class Door.



**Fig. 3.** Interlacement of visibility and navigability

As we have seen, the navigability acts as some kind of precondition for the visibility of an association, and of attributes and operations of the associated class. It has no sense to specify that an association end has public visibility (nor protected, by the way) without being navigable (see `Door.key` in the example), and in doing this the model becomes less readable. Probably, it is even better that in this case the role should remain unnamed: the association end not being navigable, the rolename has no use. In consequence, we can infer the following style rule: "when an association is navigable in only one direction, the association end which is not navigable should not have visibility other than private (or unspecified), and even the role would better remain unnamed".

## 3 Definition of navigability

### 3.1 Traversability, accessibility

The User Guide gives no definition of navigation or navigability, and the explanation given for this concept is very poor. It simply states that "given an association between two classes, it is possible to navigate from objects of one kind to objects of the other kind" [UG 143] (by the way, the concept of navigability does not apply to n-ary associations [RM 354]; n-ary

associations are out of the scope of this paper, but no doubt they deserve more attention in relation with the issue of communication). Another term that the User Guide uses as a synonym of "navigation" is "traversal". Both terms mean in ordinary discourse some kind of movement: navigate is to travel by water, to steer a course through a medium; traverse is to go or travel across or over, to move or pass along or through [Merriam-Webster, 2001]. In object orientation, the idea of movement is related to the interaction among objects, the passing of information, the sending of messages. Therefore, the novice in UML tends easily to think that navigability is the possibility of sending messages along associations, that is, a navigable association from A to B means that A objects can send messages to B objects: an idea, however, that is not clearly conveyed by the User Guide's explanations. As a reasonable simplification, this is not wrong, and even some excellent text books teach this idea explicitly (see for example [Stevens and Pooley, 2000, p. 77, p. 115]). But the truth is richer. As we will see, in UML the terms "navigation" and "traversal" are used with a meaning that is not primarily that of movement, and that only secondarily has to do with the sending of messages.

In the Standard, the term "traversal" is used in defining the metaattribute `isNavigable`: "When placed on a target end, specifies whether traversal from a source instance to its associated target instances is possible" [UML 2-24]. The explanation on the semantics of `Link` helps us something more, telling that "an opposite end defines the set of instances connected to the instance", and that "to be able to use a particular opposite end, the corresponding link end attached to the instance must be navigable", and finally that a link is used to "access" the associated instances in order to communicate with them, or to reference them as arguments or reply values in communications [UML 2-118]. In other words, "navigability" is defined more or less as "accessibility" in the context of communication, either as partner or as content of the message. The Standard says nothing more substantial about navigability, at least using this very term. Again, these few words are insufficient to form a clear concept of "navigability", a concept understandable enough to be used with confidence by modelers and tool developers.

### 3.2 Navigation expressions

Fortunately, the Reference Manual extends more on this topic, devoting an full entry in the Encyclopedia of Terms chapter. There we find that "navigability indicates whether it is possible to traverse a binary association" (nothing new until this point) "within expressions of a class to obtain the object or set of objects associated with an instance of the class" [RM 354]. And a bit further on, "navigability indicates whether a rolename may be used in expressions to traverse an association from an object to an object or set of objects of the class attached to the end of the association bearing the rolename" [RM 354]. In other words, *we can use a navigable rolename within expressions to obtain the corresponding object* (or set of objects). These expressions are called "navigation expressions" or "navigation paths", they are made up of "sequence of attribute names or rolenames" [RM 356], and they are expressed as strings in a particular language which UML does not specify (it may be OCL --the Object Constraint Language-- or another language such as a convenient programming language [UML 2-94, 3-12]). The navigation expressions are used with different purposes:

- to express constraints [RM 354];
- to map an object into a value [RM 356];
- to use the object referenced through the navigable link as argument or reply value in communications [UML 2-118];

- to communicate with the referenced object [UML 2-118].

So far, we have shown that "navigation" (or "traversal") does not mean directly "to send a message", but rather "to obtain an object", that is, *to obtain a path or reference to an object that permits handling the target object as a pseudoattribute of the source object*. As other authors put it, "navigation in OO modeling means following links from one object to locate another object" [Hamie et al., 1998]. In object orientation we want that some objects manipulate other objects, that is, invoke their public operations, get or set their public attributes, pass them as parameters of operations of other objects, and so on. Yet, *to manipulate an object, we must give that object a name*, a relative name of the target object that is valid in the context of the source object: this is exactly what a navigation expression yields, so that, we can say, *to navigate is to form the expression of a path that designates a target object (or set of objects) from a source object*.

Navigability, then, is not directly the possibility of sending messages, but the possibility for a source object to reference or name a target object, in order to manipulate it. *One of the uses* of a navigation expression is to specify the receiver object of a message, so that, indirectly, navigability results in a precondition for sending a message, but they remain nonetheless different concepts.

## 4. Consequences of navigability

### 4.1 Ability to send messages

Let's have a closer look to the relationship between navigability and communicability. Since a message may be a signal or, more frequently, the call of an operation, there are two forms of sending a message [RM 333]:

- Sending a signal from one object (the *sender*) to one of more other objects (the *receivers*)
- Calling of an operation on one object (the *receiver*) by another object (the *sender* or *caller*).

The sending of a message is the result of an action in the sender object. In the metamodel, each **Action** specifies the target of the action by means of an object set expression which resolves into zero or more instances [UML 2-120], so that, in principle, both for a signal and for the invocation of an operation the receiver may be a set of objects [RM 334] (as we have seen, however, the documentation sometimes suggests that the receiver of an operation call must be a single object, while the receivers of a signal may be multiple).

The specification of a message consists of the expression of the desired *activity* to be executed, with parameters if necessary, as well as the expression of the desired list of *addressees*, that is, the object or objects that are expected to perform the requested service. Hence, for an object to send a message to another object we need:

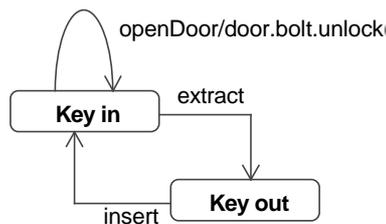
- An *visible operation* in the receiver object that may be invoked by the sender object, that is, a public operation. Alternatively, for a signal, we need a (public?) reception in the receiver object stating that it is prepared to react to the receipt of the signal (the

reception designates a signal and specifies the expected behavioral response; in the metamodel *Reception* is, like *Operation*, a subclass of *BehavioralFeature* [UML 2-109]).

- A *navigable path* from the sender to the receiver, by means of which the sender object "knows about" the receiver object and can specify it. This path is yielded by a navigation expression that is the value of the metaattribute *Action.target* [UML 2-104], which is inherited by both *CallAction* and *SendAction* [UML 2-105, 2-110].

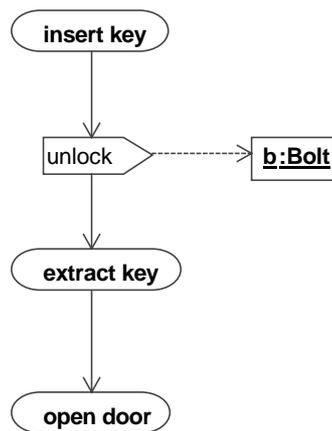
Therefore, visibility and navigability are both required for communication between objects to take place (we can put it this way: *accessibility* = *navigability* + *visibility*). *An object can communicate only with other objects it knows about*, and that have made available the desired operations (or receptions) in their interface. These ideas are rather simple, but we find that they are not clearly and concisely expressed in the official UML documentation.

In a statechart diagram the sending of a message can be expressed textually as an action attached to a transition between two states (see Figure 4), or even inside a state as an internal transition, or as entry or exit actions. In this textual notation, the path from the sender to the receiver object is expressed explicitly in the destination clause of the action expression.



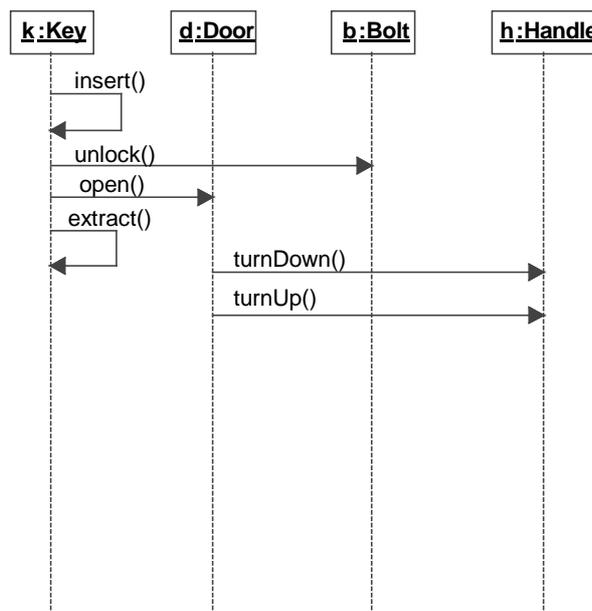
**Fig. 4.** A statechart diagram of class *Key* showing a message sent to an object of class *Bolt* as a consequence of a transition between states. The path to the receiver object is shown explicitly with a navigation expression

The Reference Manual presents an alternate graphical notation in which a dashed arrow is drawn from the transition line to a box housing the receiver's state machine [RM 420], although this notation is completely absent in the Standard. For activity diagrams there is a similar graphical notation (see Figure 5) in which "the sending of a signal may be shown as a convex pentagon (...). A dashed arrow may be drawn from the point on the pentagon to an object symbol to show the receiver of the signal" [UML 3-170]. In this graphical representation the navigation expression is not shown, and the existence of the path is implicit: the dashed arrow does not mean the path, but the sending action itself.



**Fig. 5.** An activity diagram describing the opening of a door, in which a message is sent to an object of class *Bolt*. The existence of the path to the receiver object is implicit, and its expression remains unknown

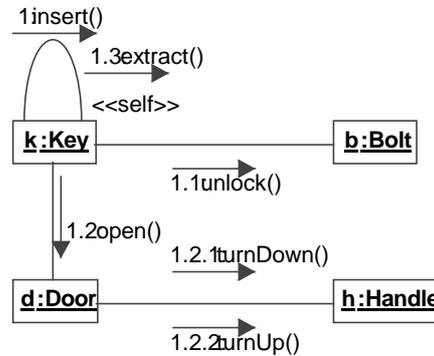
In a sequence diagram the sending of a message is represented as an arrow starting on the sender's lifeline and ending on the receiver's lifeline (see Figure 6). Note that, as in the activity diagram, the existence and expression of the path from the sender to the receiver is implicit: each arrow represents an individual message, and the path itself is not shown.



**Fig. 6.** A sequence diagram showing the interaction needed to open a door. Here the existence of the path to the receiver object is implicit, too

Finally, in a collaboration diagram we represent explicitly both the message sent and the path that the messages follows from the sender to the receiver (see Figure 7). This path is shown as a link that is said to be an instance of an association [UML 2-107, 2-118], and it may have an arrowhead to indicate that it is only one-way navigable (supposedly, because its declaring association has also this kind of navigability). The message is shown as a small arrow next to the link between instances, flowing in the given direction; there may be more than one message next to a single link, if that path is used several times in the interaction. The Standard says that "obviously such an arrow cannot point backwards over a one-way line". Well, this may seem obvious by now, since we have already settled that an object can communicate only with other

objects it knows about, but we have also shown that the ideas spread out in the official documentation are not didactic enough with respect to the implications of navigability for the sending of messages. In other words, it is true, but it is not that obvious.



**Fig. 7.** A collaboration diagram showing the same interaction as in Figure 6. The paths followed by the messages are shown explicitly in the form of links between objects<sup>3</sup>. The path between objects *k* and *b* is not an instance of any single association in Figure 3

The statement that "a link in a collaboration diagram is an instance of an association" poses two different kinds of problems. First, there may be stereotyped links that apparently do not correspond to any association in the model (for example, a link between an object and itself), but which constitute nevertheless a connection between instances along which messages can be sent [UG 210; UML 2-108]; this question is far from having been clarified, as the continuous debates demonstrate [Stevens, 2001] (see also the contributions to The Precise UML Group mailing list [Precise UML Group, 2001] during the years 2000-2001 under the subjects "links & messages", "Link as instance, tuple, path", "Sets and bags", and "Dependencies and associations", in which the author played an active role). Second, a path from the sender to the receiver may be represented by a navigation expression that is a "sequence of attribute names or rolenames" [RM 356], that is, a path would be a composition of links rather than a single link, so that obviously it cannot be an instance of a single association. Now we have a dilemma on a collaboration diagram: either we represent such a compound path as a single line connecting the sender and receiver instances (see Figure 7), or we split the path in its component links. But in this latter case we might find that there is no operation in the intermediate classes that consists simply in relaying the message until it reaches its final destination. And if we are obliged to split the path and relay the message through the intermediate objects, what is the sense of having visible associations and allow compound navigation expressions?

## 4.2 Dependency

The first consequence of navigability that we have seen is that it *enables communication*: if class A has a navigable association towards class B, then objects of class A can know about objects of class B and can send messages to them. But remember that a navigable association is not the only way to provide a navigable path towards the receiver of a message: if an object of class A knows an object of class B through a navigable association, then it could transfer this knowledge (for example, as an argument in a message) to another object, say of class C, so that

<sup>3</sup> The links  $k \rightarrow d$  and  $d \rightarrow h$  could better have a navigability marker, but the CASE tool used to draw this diagram did not allow it. What would be the navigability marker on the "link"  $k \rightarrow b$ ?

the C object could send messages to the B object. This would be represented in a collaboration diagram by a stereotyped <<parameter>> link connecting the C and B objects. We may conclude that there are multiple ways to construct navigable paths, but all of them are ultimately based on navigable associations.

The second consequence on navigability that we are going to see now is that it *creates a dependency*: if class A has a navigable association towards class B, then class A is dependent on class B. (The Reference Manual states this only in a negative form: "An association without navigability does not create a dependency from the source to the target class" [RM 355]). *Navigability means knowing*, and knowing means both communicability and dependency.

In UML, "dependency" is a term of convenience for a Relationship other than Association, Generalization, Flow, or metarelationship (such as the relationship between a Classifier and one of its Instances) [UML 2-36]. It groups very different kinds of relationships, so that a definition that can comprise all of them becomes somewhat abstract. We can find several such definitions of dependency in the official documentation:

- A semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing) [UG 23, 460].
- A using relationship that states that a change in specification of one thing may affect another thing that uses it, but not necessarily the reverse [UG 63, 137].
- A relationship between two elements in which a change to one element (the supplier) may affect or supply information needed by the other element (the client) [RM 250].
- A dependency states that the implementation or functioning of one or more elements requires the presence of one or more other elements [UML 2-37].
- A dependency specifies that the semantics of a set of model elements requires the presence of another set of model elements. This implies that if the source is somehow modified, the dependents probably must be modified [UML 2-78].
- A relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element) [UML B-6].

We want to highlight two ideas from these definitions: changes may affect, the presence is required. For a navigable association they mean: if A knows and uses B, then *the presence of B* is required for A to function, and *a change in B* may be recognized by A, so that it may have to adapt itself to the change.

Dependencies are bad for reuse, since the dependent element cannot be reused without reusing also the independent element. In terms of classes, if class A knows about class B, then it is impossible to reuse class A without reusing class B [Stevens and Pooley, 2000, p. 78]. We cannot fully eliminate dependencies between elements, but we should try to minimize them in a software project. When the associations in a model are predominantly unidirectional, *the reuse of small portions of the model becomes easier*: if class A depends on class B, then we cannot reuse A without B, but at least we can reuse B without A.

A two-way navigability in an association induces a stronger dependency than a one-way navigability: it induces two dependencies instead of only one. The consequence is clear: we should not introduce navigability unless it is actually required, or there is good reason to think it will be required in future. Sometimes it is essential to allow two-way navigability along an association, but such decisions should be individually justified, rather than being the default [Stevens and Pooley, 2000, p. 78].

## 5 Properties of navigability

### 5.1 Invertibility & bidirectionality

Now we are going to examine the property of invertibility of associations. In principle, we assume that an association is invertible when it is bidirectional (remember: bidirectional does not mean symmetric), that is, when it is navigable both ways. Invertibility would be a synonym for bidirectionality. An association that can be navigated (or traversed) both ways implies *mutual knowledge*, and hence mutual communicability and dependency. However, we are not sure that the intention of UML is this, or that UML is fully coherent with the object-oriented paradigm in this respect.

The Reference Manual is rather confusing on this topic: "An association is sometimes said to be bidirectional. This means that the logical relationships work both ways. This statement is frequently misunderstood, even by some methodologists. It does not mean that each class "knows" the other class, or that, in an implementation, it is possible to access each class from the other. It simply means that any logical relationship has an inverse, whether or not the inverse is easy to compute. To assert the ability to traverse an association in one direction but not the other as a design decision, associations can be marked with navigability" [RM 50].

According to this paragraph, it seems that:

- Logical directionality does not mean knowledge, ability to access or reference another element; logical directionality is not navigability.
- Bidirectionality, or invertibility, is a purely logical property that applies necessarily to all logical relationships, not only to some of them.
- Navigability is an implementation property that can be specified for some associations; navigability is a design issue, not an analysis issue; defining the navigability of an association only has sense at the design level.
- Invertibility is not two-way navigability; the first one is a logical concept, the second one is an implementation concept.

We have tried to make sense of the idea of bidirectionality, or invertibility, as it is expressed in the Reference Manual, but we are not satisfied with the result: this interpretation seems artificial to us. If directionality is not navigability, what is it, then? A much more natural interpretation of the intertwining between invertibility, directionality and navigability brings us to the core of the concept of association.

It is well-known that the concept of association in UML, as in its closest predecessors the Object Modeling Technique [Rumbaugh et al., 1991] and the Object-Relation model [Rumbaugh, 1987], derives from the concept of relationship in the Entity-Relationship model [Chen, 1976]. In ER modeling, the existence of a relationship between two entities is the expression that some predicate verifies in the real world, that there is a property that holds for the pair. We store the tuple (John, Acme) in the table Works-for to record the fact in the real world that "John works for Acme".

In ER modeling, relationships have no direction (of course, they are asymmetric, but this is another issue, as we have already explained). Relationships are "neutral" for the related entities, they express a fact or predicate that may be queried by someone outside the relationship itself (the database engine, for instance). Entities have no behavior, they are purely passive data, they do not perform any operation, they do not query other related entities. If "John works for Acme", this is a fact that is known neither by "John" nor by "Acme". In other words, entities are not objects. We are used to think of directionless associations (directionless is the same as bidirectional) probably because we are used to think in terms of passive data structures that are *queried from the outside*.

In OO modeling this approach does not work any more. A link between two objects, like a relationship between two entities, expresses a fact, but the linked objects themselves are responsible to inform of the existence of that fact to whoever can ask for it. A navigable association represents state information available to a class [RM 49]. The navigability of an association indicates, among other things, *who has the responsibility to tell about the state of the association* [Fowler and Scott, 1997, p. 60], who knows that state. In Figure 1 the association works-for is navigable only from Person towards Company: this means that John is able and responsible to tell whether he works for Acme or not; Acme has not this capability.

If the state of an association can be queried and updated by both ends, then the association is bidirectional; if only by one end, it is unidirectional. According to the Standard, an association with no-way navigation is normally rare or nonexistent in practice [UML 3-75]. We state it even more strongly: *a no-way navigable association has no sense at all*, because nobody would be responsible for it (except, maybe, the case of an association-class, which, having its own behavior, could be responsible itself for maintaining its state).

Now, is it really true that every logical relationship is bidirectional, invertible? This is not that clear even from the mathematical point of view. Roughly, invertibility of an association means it works both ways, it is meaningful in both directions [RM 50]. When we consider only one direction of the association, we get a mapping, a correspondence. According to Odell, a mapping assigns an object of one type to an object or set of objects of another type [Martin and Odell, 1995, p. 42]. Odell suggests also that pairs of related mappings pointing in opposite directions are inverses (in the sense of set theory), and therefore, an association consists of two inverse mappings. Other authors have shown that this is formally incorrect, but close enough to truth, and have attempted a more rigorous approach based on the category theory [Graham et al., 1997a].

Anyway, we are more interested in the object-oriented concept of invertibility, and we claim that *it makes perfect sense to say that an association is meaningful in only one direction*, and this being a logical property of the model, not an implementation property. For instance, we could consider, as Figure 3 tells us, that a Key object knows the Door object(s) it can open, but not conversely: we can make copies of the key without the door becoming aware. Why are we deprived of the possibility of specifying that a logical association is only one-way navigable?

We agree that in the first stages of analysis there is no great need to impose direction to associations [RM 49], and that navigability is mainly a design issue [Fowler and Scott, 1997, p. 61] that often cannot be decided during analysis [Stevens and Pooley, 2000, p. 78], but we do not agree that every association must be logically invertible. *The essence of association*, we claim, is *knowledge*, and knowledge can be unidirectional, not for a question of efficiency, but for a question of principle.

## 5.2 Efficiency

The Reference Manual gives a technical definition of navigation efficiency: "A binary association is efficiently navigable if the average cost of obtaining the set of associated objects is proportional to the number of objects in the set (not the upper limit on multiplicity, which may be unlimited) plus a fixed constant" [RM 356]. No doubt, efficiency is an issue to be dealt with during the design stage, and in this sense a definition based on cost of traversal is perfectly adequate.

Now, although the Reference Manual states clearly that navigation efficiency is not the defining property of navigability [RM 356], it nevertheless asserts that navigation usually carries the connotation of navigation efficiency [RM 355]. This would not be so bad if the User Guide and the Reference Manual did not suggest in some places that the navigability marker does not indicate *logical* navigability, but *efficient* navigability, and, conversely, that navigation against the navigability marker does not become *impossible*, but only *inefficient*:

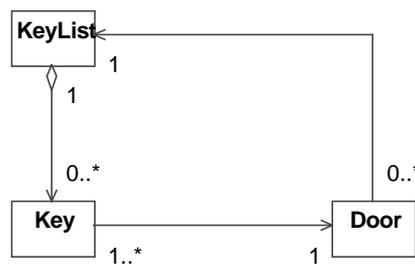
- "Specifying a direction of traversal does not necessarily mean that you can't ever get from objects at one end of an association to objects at the other end. Rather, navigation is a statement of efficiency of traversal" [UG 144].
- "A link may be used for navigation. In other words, an object appearing in one position in a link may obtain the set of objects appearing in another position. It may send them messages (called "sending a message across an association"). This process is efficient if the association has the navigability property in the target direction. Access may or may not be possible if the association is nonnavigable, but it will be probably inefficient" [RM 325].
- "Lack of navigability does not imply that there is no way to traverse the association. If it is possible to traverse the association in the other direction, it may be possible to search all the instances of the other class to find those that lead to an object, thereby inverting the association" [RM 355].
- "If an association is not navigable in a given direction, it does not mean that it cannot be traversed at all but that the cost of traversal may be significant--for example, requiring a search through a large list" [RM 357].

Apparently, then, the navigability arrow can have two different senses: logical possibility of navigation, efficiency of navigation. This ambiguity leaves the modeler in a difficult position, as any other ambiguity in the language, because it hinders the communication with other modelers and developers. The Reference Manual favors the "efficiency" version (see also [Rumbaugh, 1996a]), although this notion is absent in the Standard, where navigability is defined as sheer possibility of traversal, without any consideration for efficiency [UML 2-24]. The logical possibility of navigation is an important concept in analysis as well as in design, whereas the efficiency of navigation is relevant only for design. Therefore, in our opinion, and

against the Reference Manual, *the navigability arrow should never be used to mean efficient navigation*, especially because it makes impossible to specify an association that is not navigable at all in one direction.

If an association is logically nonnavigable in one direction, then it is certainly not invertible. This does not mean that it is absolutely impossible to know the associated object; it only means that it is impossible to know the associated object *through* the association. There may be other classes and associations in the class model that allow an alternative path that could be navigable and even efficiently navigable.

Consider the example in Figure 8. The association `Key-Door` is nonnavigable towards class `Key`. This means that a door cannot know through this very association which keys can open or close it: the association is not invertible. Fortunately for the door, for each door there exists a list of keys associated with the door, so that searching the list provides a (probably) inefficient path to its keys. The association `Key-Door` remains noninvertible itself, but we have found an alternative path through the list, and we can "invert" it indirectly (we warn that we do not like this way of speaking about "indirect inversion", since it can suggest that it is a true inversion, when it is not). Now suppose that there were no list of keys: there would be no alternative path from the door to the keys, and "inverting" the association would become completely impossible, neither directly, nor indirectly. This situation is not unreasonable, since we cannot impose that every imaginable implementation will maintain a list of the objects belonging to a certain class.



**Fig. 8.** A nonnavigable association `Key-Door` with an alternative path through another class `KeyList` and two additional associations

## 6 Notation of navigability

The Standard gives the following basic rules for the notation of navigability: "An arrow may be attached to the end of the path to indicate that navigation is supported toward the classifier attached to the arrow. Arrows may be attached to zero, one, or two ends of the path. To be totally explicit, arrows may be shown whenever navigation is supported in a given direction. In practice, it is often convenient to suppress some of the arrows and just show exceptional situations" [UML 3-75].

There is a general law in the UML notation: the suppression of some notational symbol does not mean the negation of the existence of the non-symbolized element. For instance, the suppression of the attributes compartment in a class does not mean that the class has no attributes: it only means that the modeler does not want to show them in a particular view. According to this rule, the suppression of the navigability arrow does not indicate that the association end is nonnavigable, it only leaves this property unspecified.

Unfortunately, this can lead to ambiguous models, because we cannot distinguish between "unspecified" and "nonnavigable", although modelers can avoid this ambiguity by adhering a good notation style. The Standard proposes three different styles or "presentation options" [UML 3-76]:

- *Show all arrows.* The notation of navigability is totally explicit, and the absence of an arrow indicates navigation is not supported.
- *Suppress all arrows.* The notation of navigability is totally unspecified, so that no inference can be drawn about navigation.
- *Show arrows only for one-way associations.* In this "economic" notation arrows are suppressed for associations with navigability in both directions.

A bad style would be one in which the suppression of arrows is arbitrary: some one-way associations are shown, others are not; some two-way associations are shown with two arrows, some with no arrow, and worse of all, some are shown with only one arrow. In a bad style there is no way to tell that an association end is not navigable.

Both the first and third presentation options proposed by the Standard are able to show that a certain association end is nonnavigable, although the third one is less clear, because it does not distinguish adequately between two-way and undecided navigability. Besides, in the first style the dependencies induced by navigability are expressed by the same arrows placed on association ends, while in the third style they are omitted for two-way associations. There is also a danger that unidirectional associations are considered "exceptional situations", which in our opinion is wrong (the statement that an association is bidirectional by default [UG 143; RM 355] reinforces this tendency): unidirectional associations should be considered the default, since they minimize dependencies among classes.

Therefore, we find it preferable to be totally explicit or totally unspecified, so that if a class diagram shows any navigability arrow we assume that all such arrows are shown; if it does not, we are not specifying navigability [Stevens and Pooley, 2000, p. 78]. In general, the "suppress all" style would be adequate for the first stages of analysis, whereas the "show all" would be better for a detailed analysis and for design.

As a final remark, we want to stress the point that *the notation of navigability must be used to express logical navigability, not efficient navigability*. If efficiency is important, and no doubt it is in design, a different notation must be used, such as a constraint or a stereotype.

## Conclusions

In this paper we have considered some semantic problems of associations and navigability in UML. We have tried to clarify some definitions, and we have proposed solutions for some problems, but others remain unsolved. It may happen that our comprehension of the problems is imperfect, or that our solutions are not adequate, but in any case we consider that the problems are real and the lack of clarity in the official documentation must be amended in one direction or the other.

Associations in object-oriented models, and more specifically in UML models, are not symmetrical. The main asymmetries we can find in associations are *linguistic asymmetry*,

which is basically the non interchangeability between subject and object in the verbal phrase that gives name to the association, and which is graphically expressed by the association name direction triangle; *whole-part asymmetry*, expressed by the aggregation or composition property of the association; and *communication asymmetry*, which means the direction in which knowledge can be established through the association, and which is closely related to the concepts of visibility, reference and navigation. An association can be bidirectional in the latter sense, but this does not make it symmetrical. These three kinds of asymmetry are independent, but conceptually related, so that usually they go in the same direction. Generalizations and dependencies, which are other kinds of relationships together with associations, are also asymmetric.

To "navigate" or to "traverse" an association is to obtain through the association a path or reference to the opposite object that permits handling it; in other words, *to form the expression of a path that designates a target object (or set of objects) from a source object*. Once the source object has a relative name of the target object that is valid in the source's context, the source can manipulate the target, that is, it can invoke its public operations, get or set its public attributes, pass it as a parameter of operations of other objects, and so on. Navigability, then, is the *possibility for a source object to reference or name a target object*, in order to manipulate it. The direction of navigability indicates that the object at the source end can know other objects at the target end through the association. The object that has knowledge of the association is responsible for *maintaining the state* of the association and *controlling the interaction* that can take place through it. If both ends have knowledge and are responsible of the association, then the association is said to be two-way (bidirectional), otherwise it is one-way (unidirectional). No-way navigability has no sense.

Since *navigability means knowing*, and knowing means both communicability and dependency, navigability has two main consequences. First, navigability *enables communication* from the source to the target. Visibility and navigability are both required for communication between objects to take place: an object can communicate only with other objects it knows about, and that have made available the desired operations in their interface. Navigability is so closely related to the ability of sending messages, that very often this two concepts are identified. There are multiple ways to construct navigable paths (links that are instances of associations, parameters received in previous messages, composition of links and parameters, and so on), but all of them are ultimately based on knowledge obtained through navigable associations. Second consequence, navigability *creates a dependency*, also from the source to the target. When the associations in a model are predominantly unidirectional, the reuse of small portions of the model becomes easier. This is the main argument in favor of one-way associations as a default option, instead of two-way associations as UML promotes. In any case, two-way associations cannot be completely discarded, since sometimes they are required by the nature of the problem.

We have examined two properties of associations that depend of navigability: *invertibility* and *efficiency*. According to some places of the Reference Manual, invertibility seems a logical property of an association (even of every association), different from the fact that the association is navigable both ways. Navigability would not be a logical property, but an implementation property meaning nearly the same as navigation efficiency. We consider, instead, that the logical possibility of navigation is an important concept in analysis as well as in design. *The essence of association is knowledge*, and knowledge can be unidirectional, not for a question of efficiency, but for a question of principle. Therefore, the navigability arrow should never be used to mean efficient navigation, especially because it makes impossible to specify an association that is not navigable at all in one direction.

Among the three *presentation options* recommended by the Standard, we think the best practice is using only the "suppress all" style for the first stages of analysis, and the "show all" style for a detailed analysis and for design. A connection without arrows should not be used to mean two-way navigability but only undecided navigability.

## 7 References

- [Booch et al., 1998] Booch, G., Rumbaugh, J., Jacobson, I.: *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.
- [Chen, 1976] Chen, P.P.: "The Entity-Relationship Model", *ACM Transactions on Database Systems*, 1(1):9-36, 1976.
- [Fowler and Scott, 1997] Fowler, M., Scott, K.: *UML Distilled: Applying the standard object modeling language*. Addison-Wesley, 1997.
- [Graham et al., 1997a] Graham, I., Bischof, J., Henderson-Sellers, B.: "Associations Considered a Bad Thing." *Journal of Object Oriented Programming*, 9(9):41-48, February 1997.
- [Graham et al., 1997b] Graham, I., Henderson-Sellers, B., Younessi, H.: *The OPEN Process Specification*. Addison-Wesley, 1997.
- [Hamie et al., 1998] Ali Hamie, John Howse, Stuart Kent. "Navigation Expressions in Object-Oriented Modelling." In *Proceedings of FASE'98 in ETAPS'98*. Lecture Notes in Computer Science, Vol. 1382. Springer-Verlag (1998) 123-137.
- [Henderson-Sellers and Firesmith, 1999] Henderson-Sellers, B., Firesmith, D.G., "Comparing OPEN and UML: The Two Third-Generation OO Development Approaches", *Information and Software Technology*, 41(3):139-156, February 1999.
- [Martin and Odell, 1995] Martin, J., Odell, J.: *Object-Oriented Methods: A Foundation*. Prentice Hall, 1995.
- [Merriam-Webster, 2001] Merriam-Webster OnLine Dictionary, <http://www.m-w.com/>.
- [Object Management Group, 1999] Object Management Group, *Unified Modeling Language Specification*, Version 1.4 draft, February 2001 (Version 1-3, June 1999).
- [Precise UML Group, 2001] The Precise UML Group, <http://www.puml.org/>.
- [Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenzen, W.: *Object-Oriented Modeling and Design*. Prentice-Hall International, 1991.
- [Rumbaugh et al., 1998] Rumbaugh, J., Jacobson, I., Booch, G.: *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.
- [Rumbaugh, 1987] Rumbaugh, J.: "Relations as Semantic Constructs in an Object-Oriented Language", *Proceedings of the ACM Conference on Object-Oriented Programming: Systems, Languages and Applications*, pp. 466-481, Orlando, Florida, 1987.
- [Rumbaugh, 1996a] Rumbaugh, J.: "Models for Design: Generating Code for Associations." *Journal of Object Oriented Programming*, 8(9):13-17, February 1996.

[Rumbaugh, 1996b] Rumbaugh, J.: "A Search for Values: Attributes and Associations." *Journal of Object Oriented Programming*, 9(3):6-8, June 1996.

[Stevens and Pooley, 2000] Stevens, P., Pooley, R.: *Using UML: Software Engineering with Objects and Components*. Addison-Wesley, 2000.

[Stevens, 2001] Stevens, P.: "On Associations in the Unified Modeling Language". Submitted to *UML'2001*, October 1-5, 2001, Toronto, Ontario, Canada.

[Tanzer, 1995] Tanzer, C.: "Remarks on object-oriented modeling of associations." *Journal of Object Oriented Programming*, 7(9):43-46, February 1995.

[Velho, 1994] Velho, A.V.: Attribute: A Semantic and Seamless Construct. In *TOOLS 13* (ed. B. Magnusson et al.), Prentice Hall, 1994.

**Acknowledgements.** The author would like to thank the members of The Precise UML Group for their invaluable reflections on the issues dealt with in this paper.