JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 2, No. 1, January-February 2003

Sending messages in UML

Gonzalo Génova, Juan Llorens and Vicente Palacios, Computer Science Department, Carlos III University of Madrid, Spain

Abstract

In this paper we try to clarify the issue of associations as a communication infrastructure between objects, in search for a unified view of the static and dynamic aspects of associations. Communication through associations depends on navigability and visibility, therefore the interlacement of these two concepts is examined. But first the very definition of navigability has to be settled, since the concept of navigability of associations in UML is poorly explained in the official documentation. The coherent representation of the sending of messages in different UML diagrams helps to better understand the underlying metamodel and shows that a link in a collaboration diagram is not always an instance of an association.

1 INTRODUCTION

During the last decade there has been an 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 generalizations, 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. To follow this modeling approach, object-oriented languages should also evolve and implement better this construct, since they do not provide a satisfactory implementation. The original *Object-Relation Model* [Rumbaugh 87] derived into the *Object Modeling Technique* [Rumbaugh 91], which finally was one of the main conceptual and notational sources for the *Unified Modeling Language* [OMG 01]. This view has its stronger opponents in researchers such as Brian Henderson-Sellers and other promoters of the OPEN methodology [Graham 97]. 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 99].

In object orientation the behavior of the system is defined in terms of interactions between objects, that is, message interchanges. "Sending a message" usually results in an operation invocation on the receiver (messages can have also other kinds of effects, such as object creation, object deletion, etc.). Associations are necessary for communication:

without them we would not have encapsulated objects, but isolated objects that cannot interact. Unfortunately, the attempt in UML to abstract with the same construct *both the static structure of the system and the structure of interactions between objects* is not free of problems. They are two conflicting notions of association that blend relationships between data structures with client-server relationships equivalent to inter-module procedure calls, thus confusing the data modeling and functional dependency perspectives [Simons 99]. The *data modeling perspective* inherited from ERM (entity-relationship modeling) is used to establish minimal coupling between data files and prefers bidirectional associations (when directionality is cared about), while the *functional dependency perspective* inherited from RDD (responsibility driven design) is used to establish minimal functional coupling between subsystem modules and prefers unidirectional 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 object can "look out". Nonetheless, navigability, together with visibility, has a direct impact on communication: an object can communicate only with other *objects it knows about*, that is, *objects that are connected through navigable paths that are derived from navigable associations bethween the corresponding classes* (this derivation can be via instantiation, for example, although there are other ways that we will show further on). In this paper we try to clarify the issue of associations as a communication infrastructure between objects, in search for a unified view of the static and dynamic aspects of associations. A primitive version of these ideas, together with other semantic aspects of the navigability of associations (association name direction, dependency, bidirectionality, efficiency, notation, etc.), can be found in a Technical Report by one of the authors of this paper [Génova 01].

The remainder of this paper is organized as follows. Section 2 searches for an "authorized" definition of navigability in the official documentation, which leads to the concept of "navigation expression". Section 3 examines another UML concept that is in close proximity to navigability and affects the posibility of sending messages: visibility. Section 4 proceeds by showing how the ability of sending messages through associations depends on both navigability and visibility. Finally, Section 5 examines the different forms of representing the sending of messages in UML, and extracts some consequences from this review.

Since this is a conceptual research about the official definition of navigability, our main source has been *The OMG Unified Modeling Language Specification* [OMG 01], more briefly referred to as "The Standard". This document is properly the only one which is truly "official", but there are many semantic questions that are poorly explained in it, so that we have turned to the works of the original authors of the UML in search for a clarification: *The Unified Modeling Language Reference Manual* [Rumbaugh 98], which seemed an obvious choice, and *The Unified Modeling Language User Guide* [Booch 98].

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 quote version 1.4 of the Standard, and we have checked that there have been no significant changes from version 1.3 regarding these topics.

In this paper, the three references will be cited as "UML" for the OMG UML Specification, "RM" for the UML Reference Manual and "UG" for the UML User Guide.

2 DEFINITION OF NAVIGABILITY

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].

Note that navigability is defined only for associations between two classes. 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 issues 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]. 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 00, p.77 and 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` of metaclass `AssociationEnd`: "When placed on a target end, specifies whether traversal from a source instance to its associated target instances is possible" [UML 2-23]. 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 2-113]. 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.

Navigation expressions

Fortunately, the Reference Manual extends more on this topic, devoting a 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 the rolename of a navigable association end 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 "a 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 2-88, 3-11]). 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 2-114];
- to communicate with the referenced object [UML 2-114].

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 98]. In object orientation we want some objects to 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 (nevertheless, in general the use of public attributes should be restricted to better guarantee the principle of encapsulation). 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, name or designate 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.

3 INTERLACEMENT OF NAVIGABILITY AND VISIBILITY

Navigability and visibility 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, as we can observe in two places: "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 to the given class, by any descendant of the class, or only by the class itself [UG 123] (in addition to visibility, the outside class *needs also navigability,* that is, being connected through a navigable association to the given class, as we will show in the following pages). The information given by the Reference Manual [RM 497] and the Standard [UML 2-37, 3-42] 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.

> *Well understood, any class in the same package* and with navigability *to the given class, like in the case of public visibility. In a negative form, the feature with "package" visibility cannot be used by a class outside the package, even if the outside class is associated with the class owning the feature [Stevens 01]. Thus, package visibility is more than private, but less than public. We think that the phrasing of the Standard could be improved to make clear these points.*

The visibility of an association end, which is referenced by its rolename, is defined exactly in the same way as that of a feature [UML 2-23], 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 Figure 1, for example, rolenames `bolt` and `handle` define two pseudoattributes of class `Door`, so that we can refer to `Door.bolt` and `Door.handle`. 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.*

To illustrate the interlacement between visibility and navigability, let's consider the example in Figure 1:

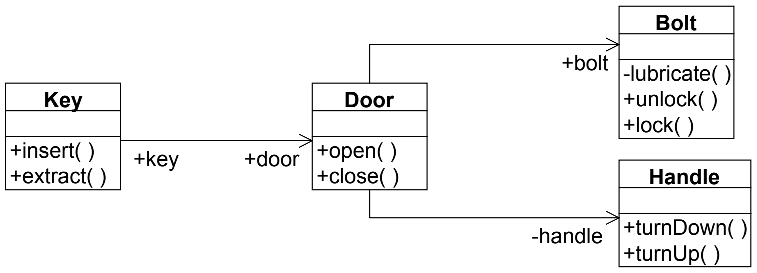


Figure 1. Interlacement of visibility and navigability. Obviously, this is only a simplified example that serves our purposes in this paper. In a more realistic model, class `Key` probably would be associated with class `Bolt`. We have omitted many details for simplicity, such as multiplicity indications.

- 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 the role were its 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: only objects of class `Door` can use this association end.

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 without being navigable, and in doing this the model becomes less readable (see `Door.key` in the example, which is uselessly public without being navigable; in fact, it has no sense to specify any kind of visibility on a nonnavigable association end). Probably, in this case the role should better remain unnamed: an association end not being navigable implies that 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 any specified visibility, and even the role would better remain unnamed”.

4 NAVIGABILITY, VISIBILITY, AND THE SENDING OF MESSAGES

Navigable path and visible operation

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 or 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.

Do not confuse the action of sending a message (which takes place in the sender object) with the action which typically occurs as a result of the reception of the message (and which takes place in the receiver object). Thus, a message usually involves two actions, one in the sender and one in the receiver.

In the metamodel (see Figure 2), the metaclass `Action` specifies *the target of the action* by means of an object set expression which resolves into zero or more instances [UML 2-99, 2-115], 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 in the preceding paragraph, 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).

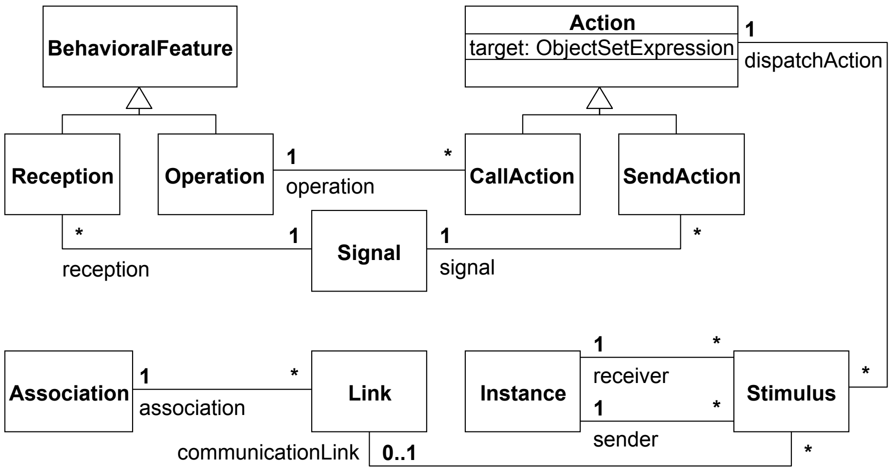


Figure 2. Metamodel of messages extracted from Figures 2-5, 2-14, 2-15, 2-16 and 2-17 in the Standard

The specification of a message, therefore, consists in the expression of the desired *action* to be executed (in fact, action or sequence of actions: "In the metamodel an `ActionSequence` is an `Action`, which is an aggregation of other `Actions`" [UML 2-99]), 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 to express the path and the action:

- The path is expressed as 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 2-99], which is inherited by both `CallAction` and `SendAction` [UML 2-100, 2-105]. Therefore, the desired list of addressees is specified by the navigation expression that results in an object set expression.

An object set expression may be obtained also as a return value of an operation. In this case we would have some kind of "nested action" to select the target instances which would execute the "main action".

- The action is expressed as a *visible operation* in the class of the receiver object that may be invoked from the class of the sender object, that is, a public operation (except in the case that the sender and the receiver belong to the same class, in which case the operation is visible without need of being public; note that in UML an object can access the private features, both attributes and operations, of other linked objects belonging to the same class). Alternatively, for a signal, we need a (public?) *reception* in the class of the receiver object stating that it is prepared to react to the receipt of the signal. Therefore, the desired action to be executed in the receiver is *directly* specified in the metamodel as an associated `Operation` in the case of a `CallAction` [UML 2-100], and *indirectly* by means of an associated `Signal` in the case of a `SendAction` [UML 2-105].

The reception designates a signal and specifies the expected behavioral response; in the metamodel `Reception` *is, like* `Operation`*, a subclass of* `BehavioralFeature` *[UML 2-104]). Unfortunately, the explanations on* `Reception` *are rather poor in the documentation. It is not sure that the reception should be public –the question mark is intentional- since the sending object really does not need to reference it when sending the signal.*

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

Representing messages in UML diagrams

How do we represent the sending of messages in UML diagrams? 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 3), 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.

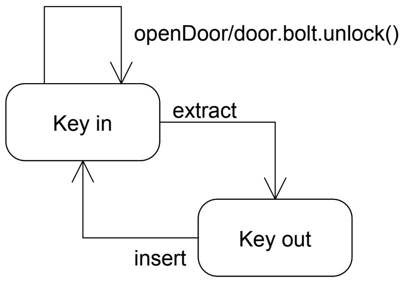


Figure 3. A statechart diagram of class `Key` showing the message `unlock()` sent to an associated object of class `Bolt` as a consequence of the transition `openDoor`. The path to the receiver object is shown explicitly by the navigation expression `door.bolt`

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 4) 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 3-166]. 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.

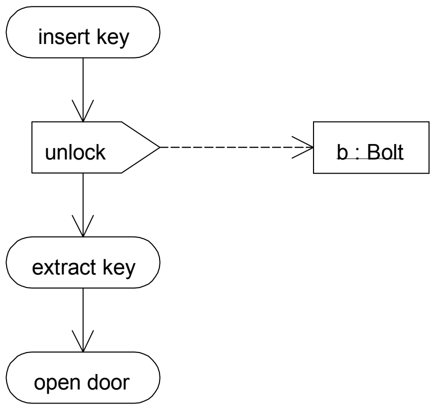


Figure 4. An activity diagram describing the opening of a door, in which the message `unlock` is sent to an object of class `Bolt`. The existence of the path to the receiver object is implicit, and its expression remains unknown

In a sequence diagram the sending of a message is represented as a *solid arrow* starting on the sender's lifeline and ending on the receiver's lifeline (see Figure 5). 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.

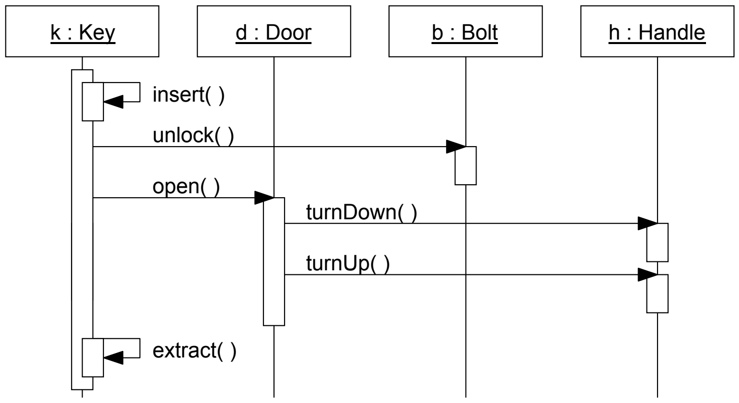


Figure 5. A sequence diagram showing the interaction needed to open a door. Here the existence of the path to the receiver object is implicit, too

Finally, in a collaboration diagram we represent explicitly *both the message sent and the path that the message follows* from the sender to the receiver (see Figure 6). This path is shown as a link that is said to be "an instance of an association" [UML 2-102, 2-113], and it may have an arrowhead to indicate that it is only one-way navigable [UML 3-115] (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.

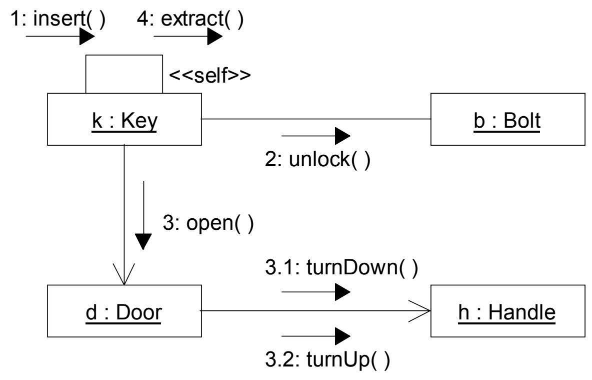


Figure 6. A collaboration diagram showing the same interaction as in Figure 5. The paths followed by the messages are shown explicitly in the form of links between objects.

The different notations we have examined are summarized in Table 1. Note that the action (the desired action to be executed) is represented always textually by the name of the sent message, which can be an operation call or a signal, whereas the path (the desired list of addressees) sometimes remains implicit and sometimes is expressed explicitly, either textually or graphically.

	Path	Action
Statechart Diagram	Navigation expression (textual)	Message name (textual)
Activity Diagram	Implicit	
Sequence Diagram	Implicit	
Collaboration Diagram	Communication link (graphical)	

Table 1. Summary of notations for the sending of messages in the different kinds of UML diagrams

Is every communication link an instance of an association?

In the example in Figure 6, the links k→d and d→h have a navigability marker according to the associations in Figure 1. On the other side, the path between objects k and b is not an instance of any single association among the associations shown in Figure 1, nor a stereotyped link, but a combination of the links k→d and d→b (the latter not shown in the diagram). What would be the navigability marker on the “link” k-b?

> *Is the path from object* k *to object* b *a link? Well, in UML 1.4, it is a link in the sense that it connects two objects in a collaboration diagram, and it supports messages; it is not a link in the sense that it is not an instance of an association. A contradiction which is the* leit motiv *of this paper.*

The Standard says that “obviously such an arrow [a message arrow] cannot point backwards over a one-way line” [UML 3-115]. Well, this may seem obvious by now, since we have already settled that an object can communicate only with other objects it knows about (see Section 4), but we have also shown that the ideas spread out in the official documentation do not present well enough the implications of navigability for the sending of messages (see Section 2). Even more, we admit the truth that a message cannot flow against link navigability, but note that link navigability may be different from association navigability. Consider the following two examples in Figures 7 and 8:

- In Figure 7 there is a one-way association from class A to class B, an object of class A sends a message to an object of class B containing itself as argument, and the B object uses the argument to send a message back to the A object. The message from the A object to the B object uses a one-way link which is an instance of the one-way association; instead, the message from the B to the A uses a one-way link in the opposite direction, which is not an instance of the association, but a stereotyped «parameter» link [Stevens 01].
- In Figure 8 there is a one-way association from class A to class B, another one-way association from class A to class C, an object of class A sends a message to an object of class C containing an object of class B as argument, and the C object uses this to send a message to the B object. Again, the message from the C to the B uses a stereotyped «parameter» link where there is no association, not even with reverse navigability [Stevens 02].

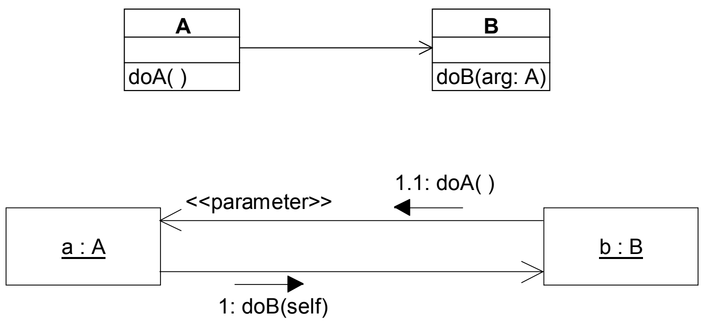


Figure 7. A collaboration diagram using a stereotyped «parameter» link against the navigability of the association A→B

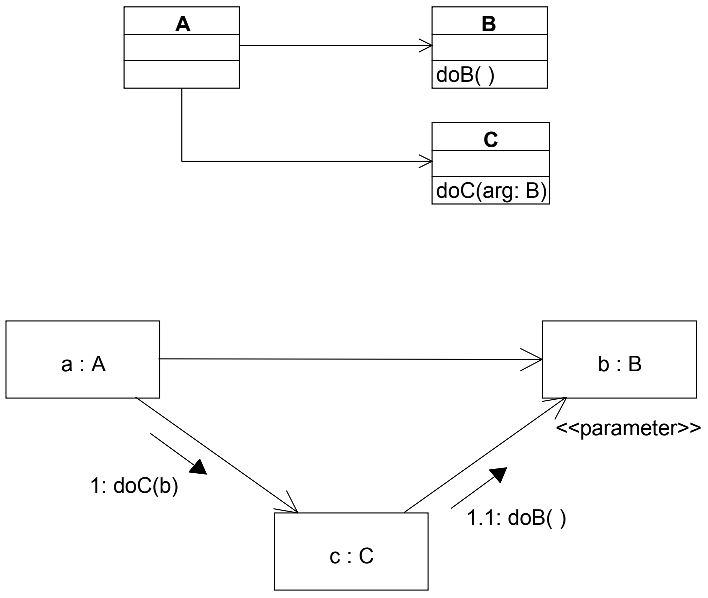


Figure 8. A collaboration diagram using a stereotyped «parameter» link without any existing association C→B

These examples show that *a link is not always an instance of an association*. In the first example, the link has its own navigability, opposed to the navigability of an association existing between the corresponding classes. In the second example, there is a navigable link where there is no association at all. Nevertheless, the links used for communication may be traced back to links that are instances of associations. We may conclude that *there are multiple ways to construct navigable paths, but all of them are ultimately based on navigable associations*.

As we have seen, 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* [UG 210; UML 2-103] that apparently do not correspond to any association in the model (for example, a link between an object and itself, see the link k-k in Figure 6; or the link between an object and the argument received in a message, see the links b→a in Figure 7 and c→b in Figure 8), but which constitute nevertheless a connection between instances where messages can be sent through. This question is far from having being clarified, as the continuous debates demonstrate [Stevens 02].

> *See also the contributions to The Precise UML Group mailing list [pUML] during the years 2000-2001 under the subjects "Links & messages", "Link as instance, tuple, path", "Sets and bags", and "Dependencies and associations", where the authors 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 *combination 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 the link `k-b` in Figure 6), 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 of 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? From the existence of stereotyped links and compound paths we conclude that *not every path or link in a collaboration diagram is an instance of an association*: there are paths that are stereotyped links, and there are paths that are combinations of links.

The Standard is rather contradictory in this respect. After stating that a message instance (a.k.a. stimulus) "uses a link between the sender and the receiver for communication", it acknowledges some *special situations in which this communication link may be missing*: "if the receiver is an argument inside the current activation, a local or global variable, or if the stimulus is sent to the sender instance itself" [UML 2-114]. The definition of a link as an instance of an association, represented in the metamodel by a mandatory association that specifies the link (multiplicity 1 on the role `Link.association`, see Figure 2) is consistent with the statement that the link is *optionally* used by a message for communication (multiplicity 0..1 on the role `Stimulus.communicationLink`) [UML 2-98]: sometimes the message uses a link (which is an instance of an association), and sometimes the message does not use any link (so no association involved). But the same document defines contradictorily *five standard stereotypes* for `LinkEnd` («global», «local», «parameter», and «self», in addition to the redundant «association») to handle those same special situations [UML 2-103]. What is the sense of defining these special links if they are not necessary, since the communication link is optional? Still worse, what is the representation of a message that is sent through a missing link in a collaboration diagram? In our opinion, the metamodel would be improved with some changes: the communication link should become mandatory for messages, the specifying association should become optional for links, and a new stereotype «compound» should be added to the set of predefined link stereotypes. Maybe the definition of a new metaclass called `Path` (or `Connection`) instead of `Link` would be better. The metaclass `Link` would be reserved for instances of associations used in object diagrams, whereas the metaclass `Path` would be used in collaboration diagrams.

A different approach could be based on Stevens' interesting distinction between static and dynamic associations [Stevens 02]. In a future work we will try to apply her ideas, which we presently do not fully support.

We hope that the coherent representation of the sending of messages in different UML diagrams and the tracing from navigation expressions to communication links has helped to better understand the underlying metamodel of messages and links. Nevertheless, the implicit representation of the path in activity and sequence diagrams may be problematic, especially when there is an ambiguity about the nature of the connecting link, which can lead developers to misunderstand a design. This ambiguity can be found even when the path is represented explicitly, but in a graphical way, as it occurs in a collaboration diagram, since the graphical notation does not convey enough information to identify uniquely the origin of the path or link.

In UML it is possible to express the association name near the path, underlined to indicate that it is an instance [UML 3-84]. However, there is no standard way to express that the path originates in a more complex navigation expression. A possible solution could be to attach a note to the link, with the expression inside.

5 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. We have searched for a definition of navigability that is missing in the official documentation, we have explored the interlacement between navigability and visibility, we have pointed out the relationship of navigability and visibility to message sending, and we have examined the different forms of representing how an object sends a message to another. 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.

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 in messages to other objects, and so on. Navigability, then, is (our definition) *the possibility for a source object to designate a target object through an association*, in order to manipulate it. This one or a similar definition should be incorporated to the Standard.

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. This idea should be expressed clearly and concisely in the Standard; the definitions of the different kinds of visibility could be improved, too.

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, combination of links, and so on), but all of them are ultimately based on navigable associations. We have shown that *a communication link is not necessarily an instance of an association*, and therefore the UML metamodel should be modified accordingly; CASE tools should take this into account, too.

Acknowledgements. The authors would like to thank the members of The Precise UML Group for their invaluable reflections on the issues dealt with in this paper. They would like to give thanks especially to Perdita Stevens for her review of a previous version of this paper; her many suggestions have served to improve it by a good deal. Among other reviewers we are grateful also to Joaquin Miller and Jennifer Tenzer.

REFERENCES

[Booch 98] Grady Booch, James Rumbaugh, Ivar Jacobson: *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[Génova 01] Gonzalo Génova. "Semantics of Navigability in UML Associations". *Technical Report UC3M-TR-CS-2001-06*, Computer Science Department, Carlos III University of Madrid, November 2001, pp. 233-251.

[Graham 97] Ian Graham, Brian Henderson-Sellers, Houman Younessi: *The OPEN Process Specification.* Addison-Wesley, 1997.

[Hamie 98] Ali Hamie, John Howse, Stuart Kent: "Navigation Expressions in Object-Oriented Modelling". In *Proceedings of FASE'98 in ETAPS'98.* Published in *Lecture Notes in Computer Science 1382*, pp. 123-137, Springer-Verlag, 1998.

[Henderson-Sellers 99] Brian Henderson-Sellers, Donald G. Firesmith: "Comparing OPEN and UML: The Two Third-Generation OO Development Approaches", *Information and Software Technology*, 41(3):139-156, February 1999.

[Merriam-Webster] Merriam-Webster OnLine Dictionary, http://www.m-w.com/.

[OMG 01] Object Management Group: *Unified Modeling Language Specification,* Version 1.4, September 2001.

[pUML] The Precise UML Group, http://www.puml.org/.

[Rumbaugh 87] James Rumbaugh: "Relations as Semantic Constructs in an Object-Oriented Language". In *Proceedings of the ACM Conference on Object-Oriented Programming: Systems, Languages and Applications*, pp. 466-481, Orlando, Florida, 1987.

[Rumbaugh 91] James Rumbaugh, Michael Blaha, William Premerlani, Frederic Eddy, William Lorensen: *Object-Oriented Modeling and Design*. Prentice-Hall International, 1991.

[Rumbaugh 98] James Rumbaugh, Ivar Jacobson, Grady Booch: *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[Simons 99] Anthony J.H. Simons, Ian Graham: "30 Things that go wrong in object modelling with UML 1.3". Chapter 17 in Haim Kilov, Bernhard Rumpe, Ian

Simmonds (eds.): *Behavioral Specifications of Businesses and Systems*, pp. 237-257. Kluwer Academic Publishers, 1999.

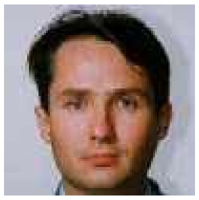
[Stevens 00] Perdita Stevens, Rob Pooley: *Using UML: Software Engineering with Objects and Components.* Addison-Wesley, 2000.

[Stevens 01] Perdita Stevens: Personal communication to the authors. August 1st, 2001.

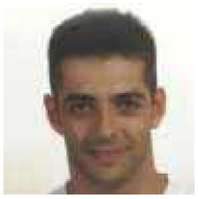
[Stevens 02] Perdita Stevens: "On the Interpretation of Binary Associations in the Unified Modelling Language", *Journal on Software and Systems Modeling*, 1(1):68-79, September 2002.

About the authors

Gonzalo Génova is currently working in his PhD in Computer Science at the Carlos III University of Madrid, Spain, where he is also a Teaching Assistant of Software Processes and Advanced Software Design. His main research subject is the interlacement of static and dynamic aspects of associations in the UML. He can be reached at ggenova@inf.uc3m.es.



Juan Llorens is Associate Professor of the Computer Science Department in the Carlos III University of Madrid, Spain, where he is the leader of the IE (Information Engineering) research group. He is also a Visiting Professor at Aland's Institute of Technology - ATL, Mariehamn, Finland. His current research involves the integration of Knowledge technologies and Software Engineering techniques towards Software and Information Reuse. He can be reached at llorens@inf.uc3m.es.



Vicente Palacios is currently working as Systems Engineer at the Carlos III University of Madrid, where he is also a Lecturer of Software Processes and Advanced Software Design. He can be reached at palacios@di.uc3m.es.