JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 7, July-August 2004

UML Associations: A Structural and Contextual View

Gonzalo Génova, Juan Llorens and José M. Fuentes
Computer Science Department, Carlos III University of Madrid

Abstract

The different kinds of communication links that can exist in an interaction among objects pose the question of whether every link *is* or *is not* an instance of an association, and whether an association must exist whenever there is a communication path between objects. The distinction between *static and dynamic associations* is not adequate to solve this problem, since in object-orientation every association has static and dynamic features, so that these two aspects do not serve to define two disjoint subtypes of association. Instead, we propose the distinction between *structural and contextual associations*, which, with an adequate redefinition of association and link stereotypes, helps to maintain the principle that every link is an instance of an association, avoiding the baseless link problem.

1 INTRODUCTION

One of the biggest difficulties in modeling with UML stems from the attempt to abstract with one construct, the association, both the *static structure* of the system and the *structure of interactions* between objects, an idea inherited from Rumbaugh's *Object-Relation Model* [Rumbaugh 87]. Unfortunately, UML does not solve the conflict between two different 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]. In particular, the role of associations as communication infrastructure between objets is not clearly explained in the UML official documentation, since apparently there can be communication links that do not belong to any existing association, against the principle that every link is an instance of an association. Some authors have tried to distinguish between two disjoint subtypes of associations to solve this problem, "static associations" and "dynamic associations" [Stevens 02], but we consider that this distinction is not adequate, since in object orientation every association has both static and dynamic features.

The remainder of this paper is organized as follows. Section 2 exposes the contradictions of the UML Standard in dealing with communication links. Section 3

Cite this article as follows: Gonzalo Génova, Juan Llorens and José Miguel Fuentes: "UML Associations: A Structural and Contextual View", in *Journal of Object Technology*, vol. 3, no. 7, July-August 2004, pp. 83-100. http://www.jot.fm/issues/issue_2004_07/article1

recalls and criticizes Stevens's distinction between static and dynamic associations. Section 4 contains the fundamentals for the distinction between structural and contextual associations, and its relationship to association and link stereotypes. Section 5 summarizes our proposal, and Section 6 applies these concepts to the representation of associations in a practical way. Finally, Section 7 contains the conclusions of this paper. We quote, by section and page numbers as usual, current version 1.5 of the UML Standard [UML], since version 2.0 is not yet approved and available to the general public. Thus, "[UML 2-110]" means "[UML] sect. 2, p. 110".

2 IS EVERY COMMUNICATION LINK AN INSTANCE OF AN ASSOCIATION?

Consider the following example. In Figure 1(a) there is a class diagram with a one-way association from class `Owner` to class `Bank`, and another one-way association from class `Owner` to class `Account`. In Figure 1(b) there is a collaboration diagram where an object of class `Owner` sends a message to an object of class `Bank` containing an object of class `Account` as argument, and the `Bank` object uses this `Account` object to send it a message: the owner object communicates its bank to close its account.

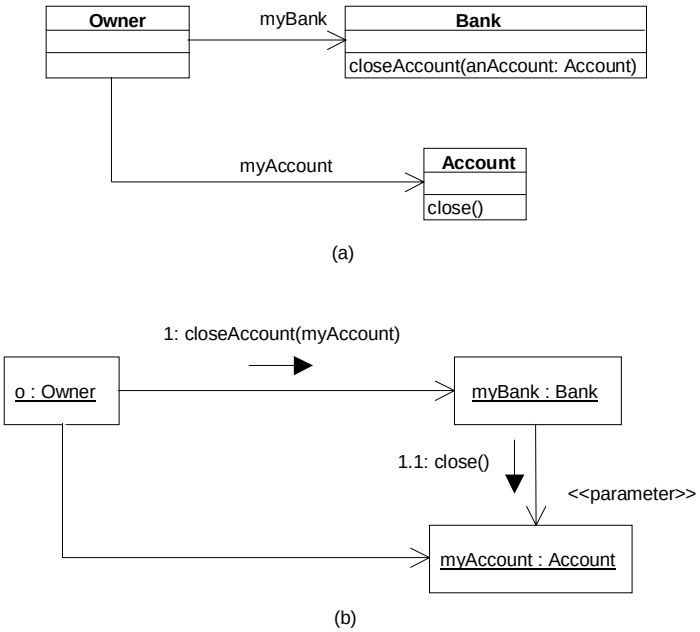

Figure 1. (a) Class diagram with classes `Owner`, `Bank` and `Account`, and two associations among them. (b) Collaboration diagram using a stereotyped `«parameter»` link without any existing association `Bank`→`Account` in the class diagram

In UML this interaction is usually modeled using a stereotyped `«parameter»` link from the `Bank` object to the `Account` object. Is this a true link, or is it only a graphical fiction? Does this link require an association between the `Bank` and `Account` classes? If not,

does this mean that the link is not an instance of any association? This question is far from having being clarified, as recent research demonstrates [Stevens 02][1].

The Standard is rather contradictory in this respect, and gives two different solutions to this problem:

- *Sometimes a message does not use a communication link*. After stating that a message instance (a.k.a. stimulus) "uses a link between the sender and the receiver for communication", the Standard 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 2-110]. Therefore, the link `myBank→myAccount` in Figure 1(b) would be a *graphical fiction*, and no association is required between `Bank` and `Account`.
- *Sometimes a link is not an instance of an association*. The Standard defines *five standard stereotypes* for `LinkEnd` (`«global»`, `«local»`, `«parameter»`, and `«self»`, in addition to the redundant `«association»`) to handle those same special situations [UML 2-100], where we find communication without associations. Therefore, the link `myBank→myAccount` in Figure 1(b) would be a *true link*, yet a link that is not derived from the existence of an association between `Bank` and `Account`, but from "other circumstances". Again, no association is required between `Bank` and `Account`.

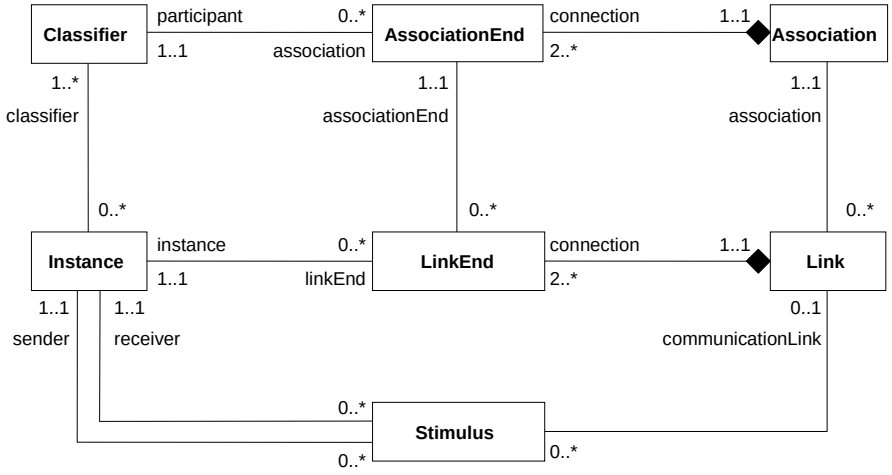

Figure 2. Metamodel of communication links extracted from Figures 2-6, 2-16 and 2-17 in the Standard

The first solution is consistent with the statement that a link *is* an instance of an association, represented in the metamodel by a mandatory `Association` that specifies

[1] 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. Other similar problems involving compound navigation expressions and stereotyped `«self»` links are described in a recent paper by the authors [Génova 03a].

the `Link` (multiplicity 1..1 on the role `Link.association`, see Figure 2), and it is also consistent with the statement that the link is *optionally* used by a message for communication (multiplicity 0..1 on the role `Stimulus.communicationLink`): 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).

The second solution breaks the principle that every link is an instance of an association, and it contradicts the first solution: if the communication link is optional, what is the sense of defining these special stereotyped links? But it is consistent with the common representation of interactions in collaboration diagrams, where a message *always* uses a link (a rule that is broken in the first solution)[2].

None of these two solutions is satisfactory. If links are optional, what is the representation of a message that is sent through a missing link in a collaboration diagram? The idea of a fictitious link does not seem a good one. In a previous paper we rejected optional communication links and supported the idea of links that are not instances of associations [Génova 03a]. However, we are not satisfied with this conclusion, since a link, like an object, is a "concrete thing" (an instance); thus, a link, like an object, requires a "type" that specifies its features; the type of an object is a class, and the type of a link should be an association that specifies, among other features, the classes of the linked objects, the navigability and changeability of the links, etc. If a link had no type, we would not be able to describe which its properties are or how it is supposed to behave[3]. Therefore, we will try to find a conceptually better solution that avoids the problem of "baseless links".

3 STATIC AND DYNAMIC ASSOCIATIONS

UML defines an association as a "semantic relationship" between classifiers [UML 2-19]. But what kind of semantic relationship? What does an association *mean*? How do we identify "something" in the problem domain or in the solution domain that should be modeled as an association? For example, which associations should exist in Figure 1(a)? Should we add an association from `Bank` to `Account` to represent the existing communication relationship?

Stevens has proposed the interesting distinction between *static* and *dynamic* associations [Stevens 02], which would lead to define two subtypes of a generic association concept in the metamodel:

- A static association expresses a *structural relationship* between classes (more generally, between classifiers), typically implemented by means of mutual references, that is, the code of each class will include at least an attribute holding

[2] This problem is hidden in sequence diagrams, which do not show links explicitly.
[3] There are some object-oriented *untyped* programming languages, such as Eiffel or Smalltalk, and even though it is completely legitimate the use of UML to model systems implemented in these languages, UML itself is strongly typed, likewise languages such as Java.

references to objects of the other class (the number of allowed references will depend on the multiplicity).

- A dynamic association expresses a *behavioral relationship* between classes, which typically implies that there will be in the implementation some kind of message interchange between objects belonging to the associated classes (thus, we can call this a *communication relationship*, too).

This distinction would be reflected in the metamodel by two new metaclasses, `StaticAssociation` and `DynamicAssociation`, which would be subtypes of the `Association` metaclass, which would then be an abstract metaclass. Moreover, Stevens proposes the use of two stereotypes, `«static»` and `«dynamic»` (more properly speaking, "keywords"), in order to distinguish them conveniently in a diagram. In the example in Figure 1(a), the association between `Owner` and `Account` would be `«static»`, the association between `Owner` and `Bank` would be both `«static»` and `«dynamic»` (or simply `«static»`), and there should be a `«dynamic»` association between `Bank` and `Account`. However, this classification of associations requires some remarks:

Static association

- An association being static does not mean that its links are fixed and do not change over the system's life. In a typical example, such as the `works-for` association between `Person` and `Company`, we can expect frequent changes of links between persons and companies. A static association does not mean the "steadiness" of links, but that each class holds within its own static structure a reference towards the other class, *regardless of the messages received*. Certainly, in a real world's problem there may be associations with links that do not change (safe for creation at the beginning and destruction at the end), but this feature is expressed in UML by the *changeability* property of an association end [UML 2-22].
- A static association exists independently of communication, but it permits message interchanges too, as we have seen in the example in Figure 1. However, static associations are insufficient to model communication links, since there exist links that are not instances of static associations.
- The existence of a static association can be deduced from the implementation, without need of executing the code to check whether it exists or not. Nevertheless, forward and reverse engineering of static associations is not at all trivial [Génova 03b].

Dynamic association

- The relationship between a dynamic association and the implementation is much more obscure. We cannot tie the existence of a dynamic association to an *actual* message interchange. First, determining the existence of a dynamic association would require the observation of the code during execution in every possible case,

taking into account the interaction with the environment, too. This would involve lots of technical difficulties, and would deprive the notion of dynamic association of any practical usefulness, since we are interested in determining the existence of such an association during the analysis and design phases (forward engineering); checking it after examination of the executing code (reverse engineering) is not enough. Second, in some cases the check would be an undecidable question by principle: that is, in general we can't tell, by inspecting the code, whether two instances belonging to two certain classes *will* exchange a message, since this is a problem of similar difficulty to Turing's Halting Machine[4]. Therefore, we must content ourselves with defining that there exists a dynamic association when there is a *potential* message exchange.

- A message exchange requires that the sender has knowledge of the receiver, that is, the sender will have some kind of reference towards the receiver. Nevertheless, we don't need to store this reference inside an attribute of the sender's class (that is, we don't need the reference to derive from a static association); instead, the receiver can be an argument or a return value of a previous message, or an object created in the course of responding a message. In any case, the implementation of a dynamic association *requires some kind of reference* as well, that is, a potential message exchange requires the existence of a certain structure in the sender's class.

In summary, static associations are not enough to model communication links, therefore we need the concept of dynamic associations, or something similar. But we cannot distinguish between static and dynamic associations by saying that "static associations are implemented as references, whereas dynamic associations are implemented as message interchanges". These definitions would not serve to distinguish two disjoint subtypes of associations: as we have seen, static associations permit message interchanges, and dynamic associations require the use of references[5].

4 THE CONTEXT OF ASSOCIATIONS

If static/dynamic is not an adequate classification of associations, how can we distinguish "normal" associations from other kinds of associations that seem relevant in modeling? UML has five predefined stereotypes for links ends (in the metamodel, `LinkEnd` metaclass) which are supposed to solve the dynamic associations issue, that is, how an

[4] Consider the families of classes A_1, A_2, ... A_n and B_1, B_2, ... B_n, where the instances of the A_i class simulate the behavior of the i-th Turing Machine, and they send a message to an instance of B_i on completion of the simulation. Since it is undecidable, in general, whether the i-th Turing Machine will halt [Turing 36], it is also undecidable whether the instance of A_i will send a message to the instance of B_i. Therefore, if we tie the existence of a dynamic association to an actual message interchange, then we cannot decide whether there exists such a dynamic association between A_i and B_i. This argument has been adapted from a similar one by Stevens [Stevens 02].

[5] Moreover, according to Stevens, we could even have associations that are static on one end and dynamic on the other end.

instance can communicate with another instance without any existing (static) association between the respective classes. The five stereotypes specify different ways in which an instance is "visible"[6] [UML 2-100]:

- «association»: the instance is visible via association.
- «global»: the instance is visible because it is in a global scope relative to the link.
- «local»: the instance is visible because it is in a local scope relative to the link.
- «parameter»: the instance is visible because it is a parameter relative to the link.
- «self»: the instance is visible because it is the dispatcher of a request.

For association ends (in the metamodel, AssociationEnd metaclass), we have the same five stereotypes[7], although their definitions are slightly different [UML 2-24]. It is worth to copy them here and compare with the preceding ones, which are rather obscure:

- «association»: specifies a real association; default and redundant option, although it can be used for emphasis.
- «global»: the target is a global value known to all elements, rather than an actual association.
- «local»: the relationship represents a local variable inside the procedure, rather than an actual association.
- «parameter»: the relationship represents a procedure parameter, rather than an actual association.
- «self»: the relationship represents a reference to the object that owns the operation or action, rather than an actual association.

The intention of the Standard in defining these five stereotypes is not very clear. On the one hand, it seems that we should have a *coherence rule*, in the sense that a link end having a certain stereotype implies the same stereotype for its corresponding association end; but the Standard does not impose this restriction. On the other hand, the four stereotypes «global», «local», «parameter», and «self» are apparently intended to give a kind of access that is not properly derived from an association, but from other circumstances, supporting the statement that there are communication links which are not instances of "normal" associations (see the wording of the first stereotype, as opposed to the others: "visible via association", "real association"; this implies that the other four stereotypes specify a kind of access that is not via a real association); that is, a stereotyped link end would correspond to no association end, not even to one having the same stereotype. But this would contradict the suggested coherence rule, and make the

[6] Note how the Standard is imprecise in using the concept of visibility in these definitions. Instead of "visible", it should say "accessible".

[7] In fact, "link stereotypes" in UML 1.4 have been replaced in UML 1.5 by "link constraints", while "association stereotypes" have been kept as they were. This merely terminological change is not significant for the issue. In the UML 2.0 draft version [UML2] association and link stereotypes have been reinterpreted in the new Connector metaclass (p. 163), but the fundamental problem of "baseless links" remains unsolved, since the type of the connector must be inferred according to a rule that is far from being simple.

stereotypes unnecessary for association ends (that is, they would be required for *link ends* only)[8].

Maybe this paradox is due to a careless writing of the Standard, rather than to a true inconsistency. We may suppose that the intention was to state that every link is an instance of an association, but there are "special" links that are not instances of normal associations, but special *implicit* associations, which do exist without need of being declared in the model, although the modeler can declare them in favor of clearness.

The use of association and link stereotypes can be related to a principle in Software Engineering which is most convenient to follow in order to understand and minimize the dependencies among different design elements, known as the *Law of Demeter*[9]. It can be summarized as follows: "Don't talk to strangers". In other words, when objects of two classes have the potential to exchange a message (there exists a link between them), the code of the sender's class should demonstrate this clearly (the link conforms to a declared association between these classes). That is, every communication link is an instance of an association (static or dynamic), and every association must be declared in the code. The *Law of Demeter* establishes that, in response to a message `m(`a_1`,` a_2`, ...` a_n`)`, an object `o` can send messages only to the following objects (we add the equivalence with the UML stereotypes):

- Other objects directly linked to `o`, that is, referenced by its attributes (`«association»`).
- The `o` object itself (`«self»`).
- A global object known to all other objects (`«global»`)[10].
- Any object received as argument in message `m`, that is, the a_1, a_2, ... a_n which are objects rather than data values (`«parameter»`).
- Objects created and destroyed by `o` in the course of its response to `m` (`«local»`)[11].

The default `«association»` stereotype is more or less equivalent to an *explicit static association*, according to Stevens's terminology. The `«self»` and `«global»` stereotypes, on the other side, would be *implicit static associations* (they exist regardless of message exchanges, or behavior). Finally, the `«parameter»` and `«local»`

[8] We have already mentioned in Section 2 another contradiction in the Standard, when it states that the communication link is not necessary in certain special situations (the same ones in which these stereotypes are defined) [UML 2-110].

[9] After the greek goddess Demeter. This law is a good-style rule in object-oriented systems design, discovered in 1987 by Karl Lieberherr and Ian Holland [Lieberherr 89], who worked in the Demeter Research Group (Northeastern University, Boston, USA). The law was later popularized in books by Booch, Budd, Coleman, Larman, Page-Jones, Rumbaugh and others.

[10] Stevens omits mentioning global objects in her exposition of the *Law of Demeter* [Stevens 02], but Lieberherr and Holland include them "for pragmatic reasons". This "global value that is known to all elements" [UML 2-24] would be equivalent to a public class with a single instance, in application of the Singleton design pattern [Gamma 94], so that it can be referenced without need of any instantiated link. Stevens does not relate the *Law of Demeter* with the stereotypes, either.

[11] Do not confuse this last case with the creation of an object which will continue to exist, linked to `o`, after completion of the operation execution; the link with this object corresponds to the `«association»` stereotype.

stereotypes, would map into *implicit dynamic associations* (they are directly related to operation invocations, or messages). Nevertheless, as we have noted before, we are not satisfied with Stevens's denominations, "static association" and "dynamic association", since every association has static and dynamic properties (that is, every association is involved in the structure and behavior of the system), therefore the static and dynamic aspects are not adequate criteria to classify associations[12]. Besides, these terms may hide the fact that every link is "dynamic", in the sense that it is created and destroyed dynamically ("static" links do change).

Instead, we propose this classification: *structural associations* (`«association»` stereotype) and *contextual associations* (`«self»`, `«global»`, `«parameter»` and `«local»` stereotypes), that is, associations that are valid depending on context[13]. We recover this way a term coined by Rumbaugh to designate certain usage dependencies between classes [Rumbaugh 98]. Table 1 summarizes both classifications.

Stereotype	Stevens (behavior)	Génova, Llorens & Fuentes (context)
`«association»`	Static association	Structural association
`«self»`		Contextual association
`«global»`		
`«parameter»`	Dynamic association	
`«local»`		

Table 1. Two possible classifications for associations

Finally, we must face the problem of links originated in navigation expressions combining several (structural or contextual) associations. According to Stevens, a link of this kind would be an instance of a *derived association*, to be considered as dynamic (or contextual, in our classification), since it is not declared in the structure of the participant classes. But note that *the use of derived associations violates the Law of Demeter* stated above, since the sender object has no direct knowledge of the receiver object (they are not connected by a "physical" link). Particularly, and in contrast with the other kinds of contextual associations, in a derived association the sender object's class does not know the receiver object's class[14], that is, it does not know its interface, what kind of messages it can receive. If the code used derived associations to send messages, it would introduce

[12] An alternative terminology with a similar sense could be *persistent associations* (the links persist between two different operation invocations of the class) against *transient associations* (the links exist only within an operation invocation). It expresses the same concept, but the terminology is even less adequate because it stresses excessively the temporal duration of the link, which can be brief or long in both cases.

[13] Another important difference between structural and contextual associations: apparently, the *arity* of contextual associations should be always 2, that is, only structural associations could be n-ary with n>2. The use of n-ary associations for communication is a rather complex issue that we must leave for later research.

[14] In fact, the "type", not the "class". We can't analyze here the confusion existing in UML between types and classes [Simons 02].

obscure dependencies between classes, which indicates an unsafe design. This does not preclude completely the use of derived associations (that is, compound navigation expressions) for communication: instead of sending the message directly to the object (or set of objects) selected by the navigation expression, you can first store the reference to the target object in a local variable (so you have to specify its class and the dependency is explicit), and then send the message to the object referenced by the local variable, so that it turns to be the case of a «local» stereotype. We will see an example in Section 6.

5 A PROPOSAL FOR ASSOCIATION STEREOTYPES IN THE UML STANDARD

In this Section we briefly propose the changes required to improve the semantics and notation of association and link stereotypes, applying the ideas developed in the preceding Section, especially the distinction between structural and contextual associations, and the *Law of Demeter*.

This proposal implies a correction in the multiplicity of the Stimulus.communicationLink role [UML 2-97] (change from 0..1 to 1..1), a new definition of the stereotypes for AssociationEnd and LinkEnd metaclasses [UML 2-24, 2-100], and a new explanation of how messages use links for communication [UML 2-110].

Associations

- Two types of association can be distinguished[15]. A *structural association* specifies a relationship between classifiers that is defined in the static structure of the associated classifiers themselves. A *contextual association* specifies a relationship between classifiers that is valid within certain contexts of the associated classifiers only.
- Every association must be declared in a well-formed model, in order to specify its features and to avoid interactions that might be inconsistent with the rest of the model.
- It is not necessary that every association appears in a class diagram. It is enough that the association is represented in the underlying model.
- The different kinds of associations are distinguished by the stereotypes applied[16].
- A derived association cannot have direct instances; in order to access objects through compound navigation expressions, the navigation expression's value must be assigned before to a local variable.

[15] This distinction is mainly conceptual. It might or might not be reflected in the metamodel by means of two different metaclasses.

[16] If the two metaclasses were added to the metamodel, then we would speak of keywords instead of stereotypes.

Links

- In a well-formed model, every link is an instance of an association.
- During the initial phases of a model's development, it is legal to represent links without specifying its association.
- A link bears the same stereotype as its association.
- Every stimulus or message instance requires a communication link.

Stereotypes

Five predefined stereotypes are proposed for the `Association` and `Link` metaclasses, the first one for structural associations, and the remaining ones for contextual associations. Note that these stereotypes are not any more applied to association and link ends, represented in the metamodel by the `AssociationEnd` and `LinkEnd` metaclasses, but to associations and links themselves. We propose a new keyword for the first stereotype, more adequate in our opinion than the current `«association»`[17].

- `«structural»`: applied to an association that specifies structural links; default and redundant option, but it can be used for emphasis.
- `«self»`: applied to a contextual reflexive association, implicit in every classifier, that specifies the implicit link that every instance has towards itself; this association has one-way navigability and 1..1 multiplicity on the target end, which bears the `selfTarget` rolename.
- `«global»`: applied to the contextual association that specifies the link between an instance and a global object known within the context of the instance; the association has one-way navigability from the instance's classifier towards the global object's classifier.
- `«parameter»`: applied to the contextual associations that specify the links between an instance and the parameters of its behavioral features; these links exist only within the context of the execution of those behavioral features, in response to messages received by the instance; the association has one-way navigability from the instance's classifier towards the parameter's classifier, and the rolename on the target end is the parameter's name.
- `«local»`: applied to the contextual associations that specify the links between an instance and the instances locally created by its behavioral features as local variables; these links exist only within the context of the execution of those behavioral features, in response to messages received by the instance; the association has ordinarily one-way navigability from the instance's classifier towards the local variable's classifier, and the rolename on the target end is the local variable's name.

[17] On the other side, it would be impossible to maintain the `«association»` keyword in this proposal, since a stereotype cannot have the same name as the metaclass it applies to [UML 2-81]. In the current version of UML, even though the keyword is inadequate and confusing, this rule is not formally violated, because the stereotype is not applied to `Association` but to `AssociationEnd`.

6 REPRESENTATION OF CONTEXTUAL ASSOCIATIONS

From this point on, we are going to examine the representation of contextual associations, in particular, how to represent these associations and their links in class diagrams, object diagrams, and collaboration diagrams.

Class and object diagrams

You will probably omit contextual associations in a *class diagram* representing the global structure of the system or subsystem, and show only structural associations, to avoid excessive clutter. Nonetheless, it is also possible, and sometimes convenient, to represent a more particular context in a class diagram, using the stereotypes defined in the previous Section to make contextual associations explicit. Rumbaugh proposes the representation of contextual associations in a class diagram as usage dependencies [Rumbaugh 98], but this presents some disadvantages. First, as we have shown in other places [Génova 01], any association induces a dependency, so that it is not clear why we should represent structural associations as proper associations, whereas contextual associations as dependencies. Second, if a contextual association is represented as a dependency, we hide its "association" character, with all its implications and features: instantiation, multiplicity, changeability, navigability, visibility, etc. Third, if the goal is not to overload the diagram, using dependencies instead of associations does not help a lot.

A good method to make explicit all the associations that affect a certain class may be the representation of a *contextual class diagram* for each class operation, in addition to the *global structural class diagram* where the class is defined. In Figure 3 we can see a structural class diagram of a banking system, showing only the structural associations connected to the `Account` class. This class defines the `number` and `balance` attributes (of `AccountNumber` and `Currency` types, considered as basic data types), and the `issueTransfer` operation, from the signature of which we can infer a contextual association with the `Account` class itself, but no other contextual associations.

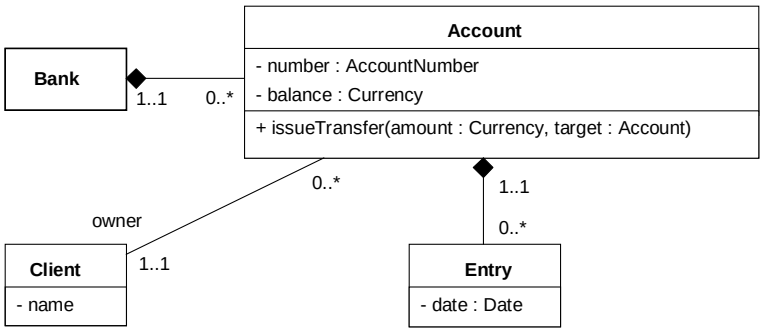

Figure 3. Structural class diagram of a bank account

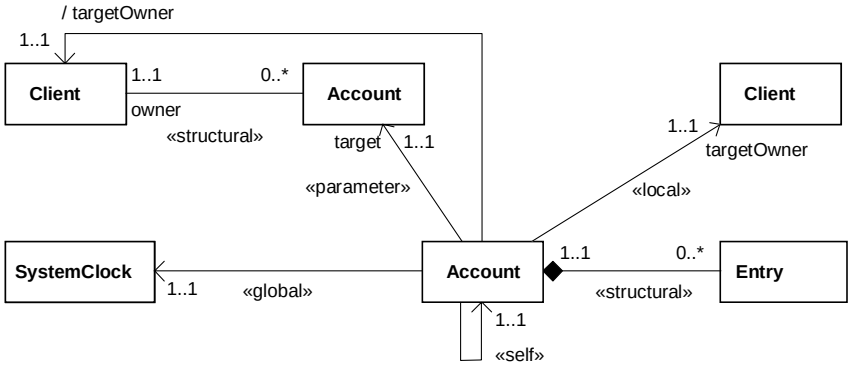

Figure 4. Contextual class diagram of the `issueTransfer` operation

The `issueTransfer` operation stores the relevant data of the transfer (current date, target account number, target account owner's name and transferred amount) as a new entry into the account's collection of entries, updates the account's balance, and notifies the target account so that it can update its data. The contextual class diagram for this operation is shown in Figure 4, where all contextual associations are made explicit. The structural association towards `Entry` is shown again, since it is relevant for the context of the operation. Besides, the following contextual associations are represented:

- A `«self»` association with the account itself.
- A `«global»` association with `SystemClock`, which will provide the date and time to be stored in the entry corresponding to the issued transfer.
- A `«parameter»` association with the `target` argument, of `Account` class, the rolename of which is the name of the argument.
- The class representing the target account shows a structural association with its `owner`, of `Client` class, so that there exists a derived contextual association that is equivalent to the `target.owner` compound navigation expression[18].
- In order to follow the *Law of Demeter*, this derived association is not directly used, but its value is read and stored in the `targetOwner` local variable, from which a `«local»` association with `Client` stems.

Figure 5 represents a different style of structural class diagram, in which, following Rumbaugh's suggestion, the contextual associations that are not structural associations at the same time are expressed as usage dependencies. Therefore, the dependencies towards `Account`, `Entry` and `Client` are omitted. In our opinion, our proposal is much more expressive and useful to understand the relationships of `Account` with other classes.

[18] The rolename is preceded by a slash (/) to indicate that is is a derived association [UML 3-93].

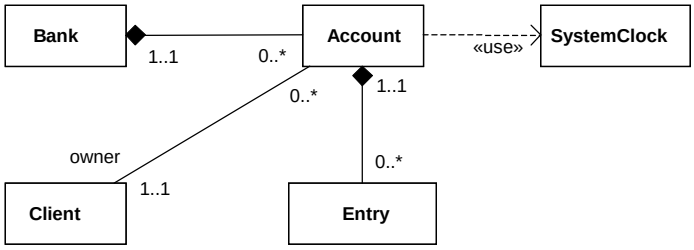

Figure 5. Structural class diagram of a bank account, with the contextual associations of the `issueTransfer` operation expressed as usage dependencies

An *object diagram* represents a concrete system situation, some kind of snapshot of the system state at a certain point of time [UML 3-35], with the involved objects and links. An object diagram, like a class diagram, is "static", in the sense that it does not represent a behavior, but a structure. This does not preclude, however, showing contextual links in an object diagram. Likewise class diagrams, if you want to represent the global system state, you will show only structural links, but if you want to represent a particular context, there is nothing against showing structural and contextual links together, with the necessary stereotypes to distinguish them. This is another reason in favor of our "contextual link" instead of Stevens' "dynamic link" terminology, since showing *dynamic* links in a *static* object diagram would seem contradictory.

Collaboration diagrams

An *instance level collaboration diagram* is similar to an object diagram with some messages exchanged in the interaction, therefore the remarks of the preceding paragraph are pertinent. In particular, even though contextual links have a special importance in a collaboration diagram, you can represent structural links, too, since these form part of the collaboration's context, and they can be used to send messages, or as message arguments or return values. Figure 6 shows a collaboration diagram corresponding to the `issueTransfer` operation, with the concrete objects, links and messages exchanged during the operation's execution. The last message in the sequence is sent to the transfer's target account, so that it updates its entries reciprocally (the arguments are omitted for simplicity).

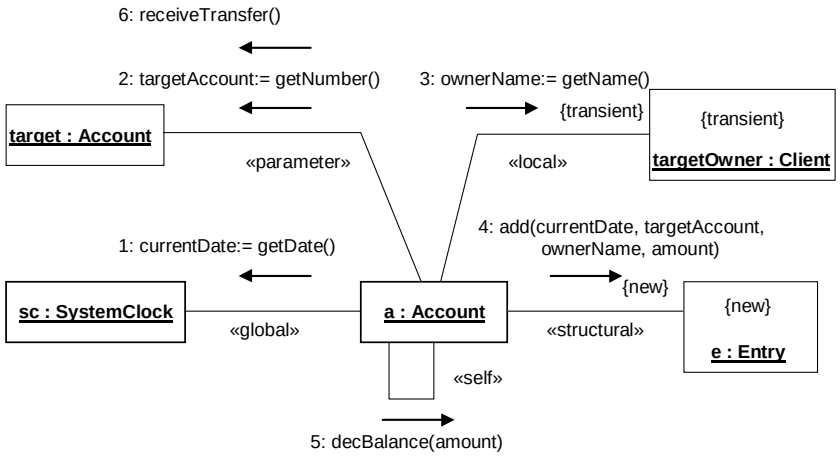

Figure 6. Collaboration diagram of the `issueTransfer` operation

Finally, a *specification level collaboration diagram* is similar to a class diagram, only it represents classifier-roles and association-roles (`ClassifierRole` and `AssociationRole` metaclasses) instead of classifiers and associations[19], and message types (`Message` metaclass) instead of message instances (`Stimulus` metaclass), so that it shows interaction patterns instead of concrete interactions. Bearing in mind the preceding remarks, it is clear that these diagrams can show both structural and contextual associations, although the latter have special relevance.

CASE Tools

CASE tools could observe all these guidelines. Probably, it is not convenient, in general, to require in a model that every link that appears in an object or collaboration diagram must correspond to an association in a class diagram. In our approach, every link is an instance of an association, but you don't need to show every association in a diagram: it is enough that these associations are represented in the underlying model, even though they do not appear in any diagram. On the other side, the declaration of `«self»` associations is superfluous, since every object has by principle a `«self»` link with itself, so that every class has a default `«self»` one-way association with multiplicity 1..1: specifying them would provide no useful information to the model. With respect to other structural and contextual links, it *is* convenient that the tool allows (even requires) the specification of the corresponding structural or contextual associations, likewise a

[19] In the metamodel, `ClassifierRole` is a subtype of `Classifier`, `AssociationRole` is a subtype of `Association`, and `AssociationEndRole` is a subtype of `AssociationEnd` [UML 2-113]. We are not going to study in detail specification level collaboration diagrams, maybe one of the obscurest and worst solved points in the Standard [Steimann 00]. Nevertheless, our contextual class diagrams are somewhat similar to specification level collaboration diagrams, but with a simpler approach that avoids role metaclasses.

class is specified for every object[20]: in this way you avoid, specially for implicit contextual associations, that the association properties remain unspecified or that the modeler specifies an interaction that is inconsistent with the rest of the model. In some contexts it may be relatively easy to suggest which these associations are. For example, if you are developing a collaboration diagram that represents the execution of a class operation, the context indicates that the receiver object has, besides the structural associations of the class, a contextual association towards each operation parameter and towards each one of its local variables.

7 CONCLUSIONS

In this paper we have examined in detail the issue of communication links, highligting some misunderstandings and conflicts in the present definition of UML (version 1.5). We have reaffirmed the principle that every link is an instance of an association, and our analysis has lead us to the distinction between structural and contextual associations, and to a new definition and application of association and link stereotypes. This distinction is not based on the static or dynamic properties of associations, since every association is (or at least may be) involved in the structure and behavior of the modeled system. Instead, our classification is based on the context in which associations are valid. The distinction is graphically expressed in diagrams using the traditional association and link stereotypes, although they are not applied to association and link ends any more, but to associations and links themselves. This work is greatly indebted to a previous proposal by Perdita Stevens, which we have extensively discussed. We hope it is not too late to consider our ideas for UML 2.0, although the new Standard is expected to appear soon.

REFERENCES

[Gamma 94] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[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.

[Génova 03a] Gonzalo Génova, Juan Llorens, Vicente Palacios. “Sending Messages in UML”. *Journal of Object Technology*, vol. 2, no. 1, Jan-Feb 2003, pp. 99-115 (http://www.jot.fm/issues/issue_2003_01/article3).

[20] In certain development phases of a software project, especially in the initial ones, it is convenient not to specify an object’s class, and in the same way the specification of a link’s association should not be mandatory. CASE tools can enable or disable this feature as it is convenient for the modeler.

[Génova 03b] Gonzalo Génova, Juan Llorens, Carlos Ruiz del Castillo. "Mapping UML Associations into Java Code", in *Journal of Object Technology*, vol. 2, no. 5, Sep-Oct 2003, pp. 135-162 (http://www.jot.fm/issues/issue_2003_09/article4).

[Lieberherr 89] Karl J. Lieberherr, Ian M. Holland. "Assuring Good Style for Object-Oriented Programs". *IEEE Software*, vol. 6, no. 5, pp. 38-48 (1989). See also Dr. Lieberherr's web page about the Law of Demeter (http://www.ccs.neu.edu/home/lieber/LoD.html).

[pUML] *The Precise UML Group* (http://www.cs.york.ac.uk/puml/).

[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 98] James Rumbaugh. "Depending on Collaborations: Dependencies as Contextual Associations". *Journal of Object Oriented Programming*, vol. 11, no. 4, pp. 5-9, July-August 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.

[Simons 02] Anthony J.H. Simons. "The Theory of Classification, Part 1: Perspectives on Type Compatibility", in *Journal of Object Technology*, vol. 1, no. 1, May-Jun 2002, pp. 55-61 (http://www.jot.fm/issues/issue_2002_05/column5).

[Steimann 00] Friedrich Steimann. "A Radical Revision of UML's Role Concept". *The Third International Conference on the Unified Modeling Language-UML'2000*, October 2-6, 2000, York, United Kingdom. Published in *Lecture Notes in Computer Science 1939*, Springer, 2000, pp. 194-209.

[Stevens 02] Perdita Stevens. "On the Interpretation of Binary Associations in the Unified Modelling Language", *Journal on Software and Systems Modeling*, vol. 1, no. 1, pp. 68-79 (2002). A preliminar version in: Perdita Stevens. "On Associations in the Unified Modeling Language". *The Fourth International Conference on the Unified Modeling Language-UML'2001*, October 1-5, 2001, Toronto, Ontario, Canada. Published in *Lecture Notes in Computer Science 2185*, Springer, 2001, pp. 361-375.

[Turing 36] Alan Turing. "On Computable Numbers, With an Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*, Series 2, Volume 42, 1936. Reprinted in M. David (ed.), *The Undecidable*, Raven Press, 1965.

[UML] Object Management Group. *Unified Modeling Language Specification*, Version 1.5, March 2003 (http://www.omg.org/).

[UML2] Object Management Group. *Unified Modeling Language Superstructure Specification*, draft Version 2.0, August 2003 (http://www.omg.org/).

About the authors

Gonzalo Génova received in 2003 his PhD in Computer Science at the Carlos III University of Madrid, Spain, where he is currently a Teaching Assistant of Software Engineering and Advanced Software Design. His main research subject is modeling and modeling languages in software engineering. He can be reached at ggenova@inf.uc3m.es.

Juan Llorens is a Professor of the Computer Science Department at 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.

José M. Fuentes obtained in 2003 his MS degree in Computer Science at the Carlos III University of Madrid. Now, he is a Teaching Assistant at the Computer Science Department of that University and has led the development of seCAKE (a new paradigm in CASE tools with reuse support developed by the dTinf Company). His current research includes software engineering, CASE tools, and knowledge management tools and methodologies. He can be reached at jmfuentes@dtinf.es.