

# Mappings, maps and tables: towards formal semantics for associations in UML 2. <sup>\*</sup>

Zinovy Diskin and Juergen Dingel

School of Computing, Queen's University,  
Kingston, Ontario, Canada  
{dingel,zdiskin}@cs.queensu.ca

**Abstract.** In fact, UML2 offers two definitions of associations. One is implicit in several Description and Semantics sections of the Spec and belongs to the UML folklore. It simply says that an association is a set of links. The other – official and formal – definition is explicitly fixed by the UML metamodel and shows that there is much more to associations than just being sets of links. Particularly, association ends can be owned by either component classes or by the very association (with a striking difference between binary and multiary associations), be navigable or not, and have some constraints on combining ownership and navigability. The paper presents a formal framework, based on sets and mappings, where all notions involved in the both definitions can be accurately explained and formally explicated. Our formal definitions allow us to reconcile the two views of associations, unify ownership for binary and multiary associations and, finally, detect a few flaws in the association part of the UML metamodel.

## 1 Introduction

Associations are amongst the most important modeling constructs. A clear and accurate formal semantics for them would provide a guidance for a convenient and precise syntax, and greatly facilitate their adequate usage. Moreover, in the context of model-driven software development, semantics must be crystal clear and syntax has to specify it in an unambiguous and suggestive way. An additional demand for clarifying the meaning of associations comes from UML2 metamodel that, in fact, is based on binary associations.

Unfortunately, the UML2 specification [7], further referred to as the Spec, does not satisfy these requirements. While complaints about informality of semantics are common for many parts of UML, for associations even their (abstract) syntax seems to be complicated and obscure in some parts. For example, the meaning of the (meta)associations *ownedEnd* and *navigableOwnedEnd* of the Association (meta)class in the metamodel is not entirely clear. More accurately, it is not easy to comprehend their meaning in a way equally suitable for

---

<sup>\*</sup> Research supported by OCE Centre for Communications and Information Technology and IBM CAS Ottawa.

both binary and multiary (arity  $n \geq 3$ ) associations. The infamous multiplicity problem for multiary associations is another point where the cases of binary and multiary associations are qualitatively different in UML (see, e.g., [3]). Even the very definition of association, in fact, bifurcates for the binary and multiary cases, though this fact is hidden in the excessively fragmented presentation of the UML metamodel via packages. A sign of distortion of the association part of the metamodel is that many modeling tools do not implement multiary associations (not to mention qualified associations - a rarity among the implemented modeling elements).

We will show in the paper that all these problems grow from the same root, and can be readily fixed as soon as the root problem is fixed. The point is that UML mixes up three conceptually and technically different sides of the association construct. In the most popular view, an association is just a collection of tuples or a table. For example, a ternary association  $A$  between classes  $X_1, X_2, X_3$  is a three-column table, whose rows are the tuples in the association and columns are the association ends. This is a purely extensional view and the roles of the classes are entirely symmetric. A more navigation-oriented view of the same association is to consider it as a triple of binary mappings

$$(1) \quad f_1: X_2 \times X_3 \rightarrow X_1, f_2: X_1 \times X_3 \rightarrow X_2, \text{ and } f_3: X_1 \times X_2 \rightarrow X_3$$

which we call *structural* (Table 1 presents it in visual form). Note that each of the structural mappings is asymmetric and has a designated target, or *goal*, class. Yet the set of three mappings  $M_S = (f_1, f_2, f_3)$  retains the symmetry of the tabular view. We will call such sets *structural maps* of associations.

When we think about implementation of structural mappings, we need to decide, first of all, which of the possible navigation directions should be most effective and which of the classes will implement it. For example, the mapping  $f_1$  can be implemented as either a retrieval operation in class  $X_2$  with a formal parameter of type  $X_3$ ,  $f_{12}^*(x:X_3): X_2 \rightarrow X_1$ , or as a retrieval operation in class  $X_3$  with a formal parameter of type  $X_2$ ,  $f_{13}^*(x:X_2): X_3 \rightarrow X_1$ . We will call such mappings *operational* or *qualified*, since UML calls formal parameters *qualifiers*. Thus, the same association can be viewed as a six-tuple of qualified mappings  $M_Q$  (see Table 1 where only three of them shown). Note that each of the qualified mappings brings more asymmetry/navigational details to its structural counterpart yet their full set  $M_Q$  retains the symmetry of the entire association; we will call such sets *operational* or *qualified maps*.

Thus, in general an association is a triple  $A = (T, M_S, M_Q)$  of mutually derivable components, with  $M_S$  and  $M_Q$  in their turn also consisting of multiple member mappings. Unfortunately, for specifying this rich instrumentary of extensional and navigational objects, the UML metamodel offers just one concept of the association *memberEnd*. For example, a ternary association consists of the total of twelve mappings while the UML metamodel states only the existence of its three ends. Not surprisingly, that in different parts of the Spec the same notion of *memberEnd* is interpreted as either a structural mapping, or a qualified mapping, or as a table column. Inevitably, it leads to ambiguities and misconceptions, only part of which was mentioned above.

In the paper we build a formal framework, where the notions outlined above together with their relationships can be accurately defined and analyzed. In a sense, we disassemble the rich intuition of the association construct into elementary building blocks and then join them together in various ways to model different views of associations. Particularly, if association is a triple  $A = (T, M_S, M_Q)$  as above, we can consider the pair  $A_S = (T, M_S)$  as its *structural projection* or *view*<sup>1</sup>, and the pair  $A_O = (M_Q, T)$  as its *operational projection/view*. The component  $T$  in  $A_S$  is understood structurally as a multi-relation (bag of tuples) while  $T$  in  $A_O$  is understood operationally as a table actually implemented in the computer.<sup>2</sup> To distinguish these higher-level views from their lower-level  $T, M_S, M_Q$  constituents, we will call the latter the association *components*.

The metamodel in Fig. 3 presents our building blocks and their relationships in a concise way. It shows a few remarkable symmetries between the components and views of associations, which is interesting to discuss (see Section 4.3). On the other hand, it forms a useful frame of reference for analyzing the UML metamodel (Section 4.4).

Formalities as such can be boring or interesting to play with, but when they are intended to model engineering artifacts, the first and crucial requirements to them is to be an adequate and careful formalization of the intuitions behind the artifacts to be modeled. We have paid a close attention to deducing our formalization from the Spec rather than from our own perception of what the association should be. To achieve this goal, we have read the Spec as carefully as possible, and discussed the possible interpretations with the experts [9, 6]. Sections 2 and 3 present the results together with an outline of some preliminary framework of main constructs. Section 4 presents an accurate formal model and sets the stage for our discussion of what is association; the culmination is in Sections 4.3 and 4.4.

## 2 What is Property? The Structural view of associations.

According to UML metamodel ([7, Fig.7.12], see our Fig. 1) an association  $A$  between classifiers  $X_1 \dots X_n$ ,  $n \geq 2$ , is an  $n$ -tuple of properties  $(f_1, \dots, f_n)$  called  $A$ 's *memberEnds* or just *ends*.

Each of the properties has its *type* [7, Figures 7.5 and 7.10] and explanations in Sect. 7.3.3 and 7.3.44 allow us to set the correspondence  $f_i.type = X_i$  for all  $i = 1..n$ . The main question is *what is the semantic meaning of Property in this definition?* The Spec says [7, Sect.7.3.44, p.121]

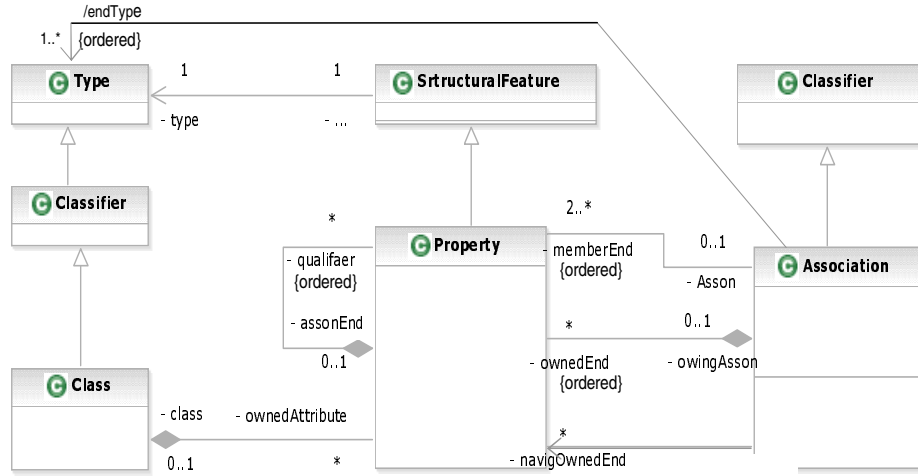
when instantiated, a property represents a value or collection of values associated with an instance of one (or, in the case of a ternary or higher-order association, more than one) type. This set of classifiers is called

<sup>1</sup> with the term “structural” meaning, arbitrarily enough, to be “implementation-free”

<sup>2</sup> In a accurate formalization, these two  $T$ 's should be different constructs; we leave it for future work and in the paper consider the notions of multi-relation and table being synonymous.

the *context* for the property; in the case of an attribute the context is the owning classifier, and in the case of an association end the context is the set of types at the other end or ends of the association.<sup>3</sup>

A natural way to interpret this definition is to consider a property in general as a mapping from some source set called the *context* (and whose elements play the role of instances “owning” the property), to a target set called the *type* of the property (whose elements play the role of values that the property takes). In particular, if the properties in question are the ends of some association, then the quote above says that each  $f_i$  is a mapping



Constraints for Association context in OCL  
(to shorten expressions we write end for memberEnd):

- (2) self.end->includesAll(self.ownedEnd) ->includesAll(navigOwnedEnd)
- (3) def: self.endType = self.end->collect(type)
- (4) if self.end->size() > 2 then self.ownedEnd = self.end<sup>a</sup>

<sup>a</sup> this is the Constraint 5 in [7, p.37],

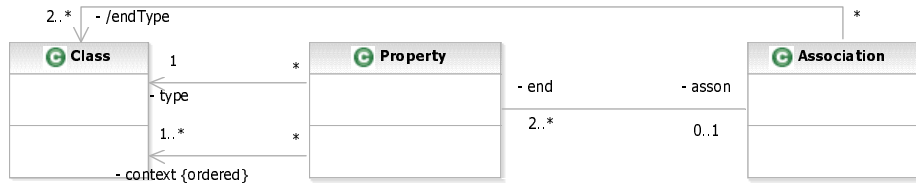
Fig. 1. A piece of UML metamodel (extracted from [7, Fig. 7.12] with additions from [7, Fig. 7.5, 7.10, 7.17]).

$$(5) \quad f_i: X_{j_1} \times \dots \times X_{j_{n-1}} \rightarrow X_i, i \notin \{j_1 \dots j_{n-1}\} \subset \{1 \dots n\},$$

<sup>3</sup> in this piece the terms “type” and “classifier” are used interchangeably and can hopefully be considered synonyms in this quote.

where the Cartesian product is the context, and the double-arrow head means that the actual target of the mapping is the set  $\text{coll}_{f_i}(X_i)$  of collections of specified (with  $f_i$ ) type (sets, bags or lists) built from elements of  $X_i$ . A special case, when the value is a single element of the target class, will be denoted by the singlearrow head, and such mappings will be called *functional* or *functions*. As a rule, we will also assume that functions are totally defined.

The left column of Table 1 shows examples of mappings of this form for association arities  $n = 2$  and  $n = 3$ . The term *multiary*, will be used generically to refer to the cases  $n \geq 3$ . Thus, an  $n$ -ary association is an  $n$ -element set of  $(n - 1)$ -ary mappings called Properties. This definition still lacks a crucial



Constraints for Association:

- (6) `def: self.endType = self.end->collect(type)`
- (7) `self1 ≠ self2 implies disjoint(self1.end, self2.end)=true`
- (8) `self.end` satisfies the constraint (inverse) in Definition 2.1(ii) <sup>a</sup>

Constraints for Property:

- (9) `self.asson.endType->includesAll(self.context)`
- (10) `self.context->size() + 1 = self.asson.end->size()` <sup>b</sup>

<sup>a</sup> constraints (7) and (8) are missed in the Spec

<sup>b</sup> constraints (9) and (10) cannot be declared in the Spec because the meta-association *context* is not there

Fig. 2. Metamodel for the *structural* view of associations

condition. Namely, we need to require that all mappings  $f_1, \dots, f_n$  are just different parts of the same association, or, as we will say, are *mutually inverse*, meaning that they all are mutually derivable by inverting/ permuting sources and targets (this condition is well known for the binary case).

Formally, this can be captured as follows. Given an  $n$ -ary mapping  $f: X_1 \times \dots \times X_n \rightarrow Y$ , its *extension*  $\text{ext}(f)$  is the collection of tuples

$$((\text{extension})) [(x_1, \dots, x_n, y) : x_1 \in X_1, \dots, x_n \in X_n, y \in f(x_1 \dots x_n) \in \text{coll}_f(Y)],$$

which is a bag if  $f$  is bag-valued.<sup>4</sup> The most natural way of presenting such a collection is to store it in a table. In fact, we have a mapping  $\text{ext}: \text{Mapping} \rightarrow \text{Table}$  sending any  $n$ -ary mapping to a  $(n+1)$ -column table recording its extension.

Now we can formulate the condition in the following way.

**2.1 Definition:** Let  $\mathbf{X} = (X_1 \dots X_n)$  be a family of classes.

(i) Any  $(n - 1)$ -ary mapping of the form (5) is called an *structural mapping* over  $\mathbf{X}$ . Its source tuple of classes is called the *context*, and the target class the *type* of the mapping.

(ii) Two or more structural mappings  $f_1 \dots f_k$  over  $\mathbf{X}$  are called *mutually inverse* if they have the same extension (up to renaming of the tables' columns)

$$((\text{inverse})) \quad \text{ext}(f_1) = \text{ext}(f_2) = \dots = \text{ext}(f_k).$$

(iii) An  $n$ -element set  $M_S = \{f_1 \dots f_n\}$  of mutually inverse structural mappings over  $\mathbf{X}$  is called a *structural map* over  $\mathbf{X}$ . In other words, a structural map is a maximal set of mutually-inverse structural mappings.

Thus, the Spec defines associations as nothing but structural maps.

**2.2 Definition: Structural view of association.** An  $n$ -ary association, *structurally*, is a set of  $n$  mutually inverse  $(n - 1)$ -ary mappings (called Properties in UML).

Precise details and terminology associated with this definition are presented in Fig. 2. This (formal) metamodel accurately describes the corresponding part of the Spec, and it is instructive to compare it with the UML metamodel in Fig. 1 (disregarding there, for a while, the ownership aspect).

**2.3 UML metamodel of associations in the light of formalization, I.** We note that the Spec misses two important constraints on associations: disjointness, (7), and being inverse, (8) in Fig. 2 (though, of course, implicitly they are assumed). Note also that our formal metamodel does not require the set of ends to be ordered. Indeed, ends are analogous to labels in labeled records: ordering is needed when there are no labels for record fields (and means, in fact, using natural numbers as labels). Thus, ordering of meta-association *memberEnd* required in the UML metamodel is redundant.

Finally, the most serious (and even somewhat striking) distinction is that the meta-association *context* is absent in the UML metamodel. As we have seen, the Spec does talk about this fundamental component of the association constructs, yet formally it is not entered into the metamodel. Is it hidden or lost in the long package merge chains in which the UML metamodel is separated? Note that even if the association *context* can be derived from other parts of the metamodel, its presence in Figure 7.12 of the Spec, the main part of the UML association metamodel, is essential since (i) without *context* we cannot formulate important structural constraints (9,10) in Fig. 2, and (ii) maybe even more important, its absence essentially decreases understandability of the metamodel.

---

<sup>4</sup> If  $f$  is list-valued, we can either disregard the ordering information by considering the underlying bag, or consider the extensional set to be partially-ordered.

### 3 A battle of ownerships: The operational view of associations

In this section we consider that part of the UML association metamodel, which specifies ownership relations between Classes, Properties and Associations. The Spec considers two specific subsets of the set  $A.memberEnd = \{f_1..f_m\}$  of association ends: the set of ends *owned* by the association,  $A.ownedEnd \subseteq \{f_1..f_m\}$ , and the set of *navigable owned ends*,  $A.navOwnedEnd \subseteq A.ownedEnd$ . Unfortunately, there is no direct explanation of the meaning of these two notions and we need to extract it from semi-formal considerations in Sect. 7.3.3 and 7.3.44.

Since for multiary association (when  $n \geq 3$ ), the notions of *memberEnd* and *ownedEnd* coincide due to the constraint (4) in Fig. 1, we have to consider binary associations to understand the difference.

It appears that the Spec assumes (though does not state it explicitly) that if an end, say,  $f_1$ , is not owned by the association,  $f_1 \notin A.ownedEnd$  then it is owned by its source classifier  $X_2$ ,  $f_1 \in X_2.ownedAttribute$ . In this case,  $f_1$  is considered to be an  $X_2$ 's attribute [7, p.121]. What is, however, the meaning of the other end,  $f_2$ , owned by  $A$ ?

We have two subcases: (+) when  $f_2$  is a navigable end,  $f_2 \in A.navOwnedEnd$  and (-) when it is not. In case (+), the association is navigable from  $X_1$  to  $X_2$  (Sect.7.3.3, p.36) and hence we have a mapping  $f_2: X_1 \rightarrow X_2$  yet  $f_2$  is not an attribute of  $X_1$  (otherwise it would be owned by  $X_1$  rather than  $A$ ). The only reasonable explanation that we could find for this situation is that mapping  $f_2$  is not supposed to be stored in the instantiations of  $A$  yet it can be derived from other data (by submitting and computing a corresponding query). Namely, we assume that mapping  $f_1$  is actually stored (with the instantiations of classifier  $X_2$  as its attribute) while  $f_2$  can be derived from  $f_1$  by taking the inverse. Strictly speaking, in case (+) association  $A$  consists of only the end  $f_1$  (stored and owned by  $X_2$ !) but can be augmented with the other end,  $f_2$ , by a suitable derivation procedure (of inverting a mapping).

Case (-): the end  $f_2$  is owned by  $A$  and is *not* navigable. The Spec says that in this case  $A$  is *not* navigable from  $X_1$  to  $X_2$  (Sect.7.3.3, p.36) and, hence,  $f_2$  cannot be considered as a mapping. Then the only visible role of  $f_2$  is to serve as a place-holder for the respective multiplicity constraint,  $m_2$ . We can consider this situation as that semantically association  $A$  consists of the only end/mapping  $f_1: X_2 \rightarrow X_1$ , whose extension (graph, table) is constrained by a pair of multiplicity expressions  $C = (m_1, m_2)$ . In this treatment, the second end  $f_2$  appears only in the *concrete* syntax as a way to visualize the second component of a single constraint  $C = (m_1, m_2)$  rather than have any semantic meaning.

We can also reformulate this situation by saying that some constraint to mapping  $f_1$  is specified by setting a constraint  $m_2$  to a mapping  $f_2$  derived from  $f_1$ .<sup>5</sup> In such a formulation case (-) becomes close to case (+). In both cases,

---

<sup>5</sup> Consider, for example, a database schema for storing objects of class *Person* with an attribute *birthDate*, with an additional constraint that for any object its age should

association  $A$  consists, in fact, from the only end  $f_1$  (owned by  $X_2$ ) while the second end is derivable rather than storable and serves for (i) specifying the  $m_2$ -half of the multiplicity constraint to  $A$  and, (ii, optionally) for navigation from  $X_1$  to  $X_2$ .

Thus, with help of implementation concepts, we were able to explain the mixed ownership cases (+) and (-). To be consistent, now we need to reconsider the case when the both ends are owned by the association. Thinking along the lines we have just used, we conclude that in this case we deal with a situation when information about the association is stored somewhere but not in the participating classifiers (otherwise the ends were attributes owned by the classifiers). Hence, to make the ends derivable mappings we need to have a source of storable data for deriving the mappings, and the classifiers  $X_1, X_2$  cannot be used for that.

A reasonable idea is to introduce onto the stage a new set, say,  $R$ , immediately storing links between instances of  $X_1, X_2$ , that is, pairs  $(x_1, x_2)$  with  $x_1 \in X_1, x_2 \in X_2$ , together with two projection mappings  $p_i: R \rightarrow X_i$ . In other words, we store the links in a table  $T = (R, p_1, p_2)$  with  $R$  the set of rows and  $p_1, p_2$  the columns so that if for a row  $r$  we have  $r.p_1 = o_1 \in X_1$  and  $r.p_2 = o_2 \in X_2$ , it means that the row stores the link  $(o_1, o_2)$  (see Table 1). We can advance this interpretation even further and identify  $R$  with  $A$  and projection mappings  $p_i$  with  $A$ 's ends  $f_i, i = 1, 2$ . This new view of associations (though may look somewhat unusual for the UML style) possesses a few essential advantages:

1. It perfectly fits in the UML idea that an association is a classifier whose extension consists of links.
2. It is generalized for  $n$ -ary associations in a quite straightforward way: just consider  $R$  with a family of  $n$  projections  $p_i: R \rightarrow X_i$ , which automatically makes  $R$  an  $n$ -column table, i.e., a collection of  $n$ -ary tuples/links.
3. A property is again a mapping and, moreover,
  - 3.1 the classifier owning the property is again the source of the mapping,
  - 3.2 the type of the property is the target of the mapping as before.

This interpretation brings an essential unification to the metamodel, and possesses a clear semantics well-explored in the relational data modeling. It also shows that the “ownership-navigability” part of the UML metamodel implicitly switches the focus from the analysis/structural view of association (Definition 2.2) to more technical (closer to design) view, where the modeler begins to care about which parts of the association will be stored, and which will be derived (with an eye on how to implement that later). We will call this latter view of associations *operational*.

The UML metamodel attributes the operational view to binary associations only (see Constraint (4) in Fig. 1). It appears to be an irrelevant restriction as in the next section we show that the operational view, including all nuances of

---

be more than 18. This is a reasonable constraint to class *Person* but it is set by constraining some information (the attribute *age*) that is not in the schema yet can be derived from it.

ownership relations, can be developed for the general case of  $n$ -ary associations as well.

## 4 Formal model for UML associations: separation and integration of concerns

In this section we build a formal framework for accurate definition of the concepts that appeared above. We also introduce a new, and important, actor on the stage: qualified or operational mappings, which are an analog of attributes for multiary associations. It is this actor whose improper treatment in the UML metamodel leads to a striking difference between binary and multiary associations.

### 4.1 Basic definitions and conventions.

Our first concern is to set a proper framework for working with names/labels in labeling records and similar constructs.

**4.1.1 Definition: Roles and contexts.** Let  $\mathcal{L} = \{\ell_1 \dots \ell_n\}$  be a *base* set of  $n$  different labels/symbols called *role names*.

(i) A *role* is a pair  $\ell:X$  with  $\ell \in \mathcal{L}$  a role name and  $X$  a class. A (*n association*) *context* is a set of roles  $\mathbf{X} = \{\ell_1:X_1, \dots, \ell_n:X_n\}$  such that all role names are distinct (while the same class may appear with different roles). We also write  $X_\ell$  for the pair  $(\ell:X)$ . Cardinality of the base set is called the *arity* of the context. For example, the set  $\{course:Subject, student:Person, professor:Person\}$  is a ternary context.

(ii) We use the term class and set interchangeably. For our goals in this section, classes are just sets of elements (often called objects). We write  $\bigcup \mathbf{X}$  for  $\bigcup \{X_\ell \mid \ell \in \mathcal{L}\}$ . We also remind the reader our convention about distinguishing general and functional mappings (presented in sect.2 immediately below formula (5)).

(iii) All our definitions will be parameterized by some context  $\mathbf{X}$ . We will say that the notions are defined *over the context*  $\mathbf{X}$ .

**4.1.2 Naming convention.** Below we will deal with a few constructs, whose definitions need some choice of names (e.g., for columns in tables to be built, or for formal parameters of operations). Generally speaking, this choice is arbitrary. However, we will see that when our constructs work within some context  $\mathbf{X}$ , the items to be named are immediately semantically related to the roles from  $\mathbf{X}$ . In this case, we will use the names from  $\mathcal{L}$  rather than introducing new ones.

**4.1.3 Definition: Links, products, relations.**

(i) A *link* over  $\mathbf{X}$  is a functional mapping  $r: \mathcal{L} \rightarrow \bigcup \mathbf{X}$  s.t.  $r(\ell) \in X_\ell$ . The set of all links over  $\mathbf{X}$  will be denoted by  $\prod_{\mathcal{L}} \mathbf{X}$  or just  $\prod \mathbf{X}$ . If  $\{(\ell : X) \mid \ell \in \mathcal{K}\}$  is a sub-context of  $\mathbf{X}$  for some  $\mathcal{K} \subset \mathcal{L}$ , we will write  $\prod_{\mathcal{K}} \mathbf{X}$  for the set of the corresponding sub-links.

(ii) A (*multi*)*relation* over  $\mathbf{X}$  is a (multi)set of links over  $\mathbf{X}$ . If  $R$  is a multi-relation,  $R!$  will denote  $R$  with duplicates eliminated, thus,  $R! \subset \prod \mathbf{X}$ . Note that

$R$  can be written down as a table whose column names are role names from the base set and rows are links occurring in  $R$ . Since each column name must be assigned with its domain, actually column names are pairs  $\ell:X$ , that is, roles.

**4.1.4 Construction: Tables vs. relations.** In the relational data model a table is viewed as a collection of rows (links). However, it is possible to switch the focus from rows-links to columns-roles and consider the same table as a collection of columns. Each column  $\ell:X$  gives rise to a functional mapping  $\llbracket \ell \rrbracket : R \rightarrow X$ ,  $\llbracket \ell \rrbracket(r) \stackrel{\text{def}}{=} r(\ell)$ . Note that  $R$  is always a set but it may happen that two different rows  $r \neq r'$  store the same link if  $\llbracket \ell_i \rrbracket(r) = \llbracket \ell_i \rrbracket(r')$  for all  $\ell_i \in \mathcal{L}$ .

(i) A *table* over  $\mathbf{X}$  is an  $n$ -tuple  $T = (p_1 \dots p_n)$  of functions  $p_i : R \rightarrow X_i, i = 1 \dots n$  with a common source  $R$  called the *head*. Elements of  $R$  will be also called *rows*, and functions  $p_i$  *columns* or, else, *projections*, of the table. We will often make the head explicit and write a table as an  $(n + 1)$ -tuple  $T = (R, p_1 \dots p_n)$ .

(ii) Following our naming convention, we will identify projections  $p_i$  with semantics of the roles in the context,  $p_i = \llbracket \ell_i \rrbracket$ .

**4.1.5 Definition: Mappings and maps over a context.**

(i) An *structural* mapping over  $\mathbf{X}$  is a mapping of the form  $f : \prod_{\mathcal{K}} \mathbf{X} \rightarrow X_\ell$ , where  $(\mathcal{K}, \{\ell\})$  is a partition of  $\mathcal{L}$  (with the second member being a singleton). The sub-context  $\{\ell : X_\ell \mid \ell \in \mathcal{K}\}$  is called the *source context* of  $f$ .

(ii) A *qualified* or *operational* mapping over  $\mathbf{X}$  is a mapping of the form  $g : X_{\ell'} \rightarrow \llbracket \prod_{\mathcal{P}} \rightarrow X_{\ell''} \rrbracket$ , where  $(\{\ell'\}, \mathcal{P}, \{\ell''\})$  is a partition of  $\mathcal{L}$ . The square brackets denote the set of all structural mappings of the form inside the brackets.

The set  $X_{\ell'}$  is the source, the roles in  $\mathcal{P}$  are parameters and  $X_{\ell''}$  is the target (goal) set. If  $\mathcal{P} = \{j_1 \dots j_k\}$ , in a standard programming notation the mapping could be written as  $g(\ell_{j_1} : X_{j_1}, \dots, \ell_{j_k} : X_{j_k}) : X_{\ell'} \rightarrow X_{\ell''}$ . We will call the sub-context  $\mathbf{P} = \{X_\ell \mid \ell \in \mathcal{P}\}$  the *parameter context* or *qualifier*.

(iii) Given a structural and operational mappings  $f$  and  $g$  as above, we say that  $g$  *implements*  $f$  if  $\ell'' = \ell$  (and hence  $\ell' \in \mathcal{K}$ ). This is nothing but a well-known Curry construction (see, e.g., [4]), and we will also call the passages from  $f$  to  $g$  and back *Currying* of  $f$  and *unCurrying* of  $g$ .

The left and middle columns of Table 1 show how it works for the cases of  $n=2$  and  $n=3$ . For the row  $n = 2$ , Currying is trivial. For the row  $n = 3$ , Currying will produce six operational mappings: two for each of the structural mappings. We show only three of them. It is easy to see that any  $n$ -ary structural mapping has  $n$  operational/qualified implementations.

(iv) An *operational/qualified map* over  $\mathbf{X}$  is a set  $M_O$  of  $n(n - 1)$  qualified mappings with the same *unCurry*-value in structural mappings. In other words, such a map is the set of all qualified mappings generated by some structural mapping. The members of an operational map over  $\mathbf{X}$  represent all *direct* navigational functionalities between participating classes.

(v) To ease comparison of our formal constructs with those defined in UML and avoid terminological clash, we will call the members of a qualified map *legs* while members of a structural map Definition 2.1 will be called *arms*.

Let  $f : \prod_{\mathcal{K}} \mathbf{X} \rightarrow Y = X_j$  be a structural mapping as above. Its extension can be presented as a table  $T = \text{ext}(f)$ . However, during this passage the information

Table 1. Three components of associations

	Structural: maps (sets) of mappings	Operational: maps (sets) of operations (parameterized mappings)	Extensional (tables)
n=2			
n=3			
n=4	...	...	...

about which of the columns of the table corresponds to  $f$ 's target is lost. Any other mapping with the same extension will result in the same table, and conversely, by looking table  $T$  up in different “directions”, we will obtain  $n$  different structural mappings including  $f$ . We remind the reader that we have called such sets of structural mappings *structural maps* (Definition 2.1(iii)). Thus, a table is an exact extensional representation of maps rather than mappings.

**4.1.6 Construction: Adding navigation to tables.** We can enrich tables with “navigational” information about the mapping generated the table if the corresponding column name will be marked (say, by a star). Similarly, if a table stores the extension of a qualified mapping, we can keep this information by marking the two corresponding columns. In this way we come to the notions of:

- (i) a *star-table* is a table with one column specially designated and called the *goal*;
- (ii) a *double-star table* is a star-table with one more column designated/marked as the *source*. In programming terms, this column could be also named *self*.

## 4.2 Formalization of ownership in the UML metamodel of associations.

As it was noticed in sect.3, the ownership meta-associations in the UML metamodel are related to possible implementations of structural associations. The latter can be implemented either by a table, or/and by any number of qualified mappings between the participating classes. Which implementation is most suitable depends on which navigation directions need to be implemented efficiently.

**4.2.1 Definition: Operational view of associations.** *Operationally*, an association over  $\mathbf{X}$  is an triple  $A = (M_O, T, B)$  with  $M_O$  an operational map of

qualified mappings over  $\mathbf{X}$ ,  $T$  their common extension table, and  $B$  a non-empty subset of the set  $M_O \cup \{T\}$ , whose elements are called *basic* while other elements of  $M_O \cup \{T\}$  are called *derived*. The intuition is that the elements of the set  $M_O \cap B$  are to be implemented as retrieval operations of the corresponding classes (their attributes in the binary case); the classes then own these elements. If also  $T \in B$ , then the extension is to be really stored in some table  $T$ . The elements formally called “derived” can be indeed derived from the basic elements (by say looking up the extension table in the required direction, and the extension table can be derived by recording the input-output pairs).

The rightmost part of Fig. 3 present the metamodel of this definition.

**4.2.2 Remark: uniqueness constraints.** It was proposed in [5], that even if the extension table contains duplicates and hence all qualified mappings from  $M_O$  are bag-valued, it may be useful for navigational purposes to choose for some of them their versions with eliminated duplicates. Then, operationally, an association over  $\mathbf{X}$  is defined to be a quadruple  $A = (M_O, T, B, U)$  with the triple  $(M_O, T, B)$  as above and  $U \subset M_O$  is the set (perhaps, empty) of those members that we have chosen to consider with eliminated duplicates. Details and a thorough discussion can be found in [5].

### 4.3 The metamodel: playing LEGO blocks with associations

Figure 3 presents the metamodel of the notions and transformations we have defined above. All meta-classes in the model are parameterized by the association’s arity  $n$ . It allows us to capture numerous important size constraints (like constraint (10) in Fig. 2) by stating the corresponding multiplicities. We believe that it would also be useful for the UML metamodel as well.

In the vertical direction, the metamodel consists of two parts: the upper half presents the extensional, or tabular, view/component of associations, the lower half shows the procedural, or map-based, view. Each of the parts is based on the corresponding structural foundation: the role context for the maps, and the column context for the tables. These two context are in one-one correspondence via the *semantics-name* meta-association, see Construction 4.1.4(ii), and it is our conjecture that in a deeper formal setting they could be unified into a single notion.

There is also a nice parallelism between the two parts in their treatment of navigability as the consecutive augmentation of the respective constructs with additional “navigational” information (what is declared to be the source and the target of the corresponding mapping). To underline this parallelism, we have denoted the (meta) associations “source context” for structural mappings, and “parameter context” for operational mappings, by *context\** and *context\*\** respectively. This “addition of navigability” is governed by one-to-many associations in both parts. One  $n$ -column Table generates  $n$  starTables, and each starTable generates  $(n - 1)$  doubleStarTables, and similarly for Maps, structuralMappings and operationalMappings. The two parts are tightly connected by vertical meta-associations *ext-lookUp* and diagonal meta-associations (shown

in dashed line) derived by the respective compositions of horizontal and vertical meta-association ends.

In the horizontal direction, the metamodel also consists of two parts: the structural view of associations (the left half) and the operational view of associations (the right half). These two views are also tightly connected by horizontal meta-associations of *Currying-unCurrying* and (*set self-column*) – (*forget self-column*).

In fact, our metamodel presents a toolbox of blocks for building different views/notions of associations. For example, structurally an association is a pair  $A_S = (M_S, T)$  with  $M_S$  a map of mutually inverse structural mappings and  $T$  the table representing their common extension ( $A_S$ 's collections of links). Operationally, an association is a triple  $A_O = (M_O, T, B)$  with  $M_O$  a map of mutually inverse operational/qualified mappings,  $T$  the extension table and  $B$  sorting the elements of  $M_O \cup \{T\}$  into basic-derived. We say that  $A_O$  *implements*  $A_S$  if they have the same extension  $T$  (and hence, mappings in  $M_O$  are Currying versions of mappings in  $M_S$ ). Extensionally, an association is a table  $T$ , and procedurally, it is a pair of maps  $(M_S, M_O)$ . We can consider an integrated notion of association by defining it as a quadruple  $A = (M_S, T, M_O, B)$ . Then all the views mentioned above are indeed views, that is, different projections/parts of the whole construct.

#### 4.4 UML metamodel in the light of formalization, II

It is instructive to compare our formal model of associations specified in Fig. 3 with the UML model (Fig. 1). Our formalization clearly shows three components of the association concept: extensional, structural and operational (Table 1). They all have the same underlying structure: a host object (a table/ structural map/ qualified map) holds a number of member mappings (columns/ arms/ legs respectively). Though these components are closely related and, in fact, mutually derivable, they consist of *different* elements: a simple calculation shows that an  $n$ -ary association  $A$  consists of the total of  $m_A = n + n(n - 1) + n = n(n + 1)$  mappings (columns, arms and legs) plus one set of links (the head). Note that all these association's elements appear in one or another way in different Semantics and Description sections of the Spec, and are used for defining associations' (meta)properties like ownerships, navigability, multiplicity. However, as formally defined by the UML metamodel, an  $n$ -ary association  $A$  consists of only  $m_A(UML) = n$  elements, its *memberEnd* Properties (UML's analog of mappings). Thus, UML metamodel offers only one package of terms and constructs to cover three packages of terms and constructs. In Fig. 3, we have pointed out UML counterparts of our formal constructs by their names in square brackets, which makes the shortage of constructs in the UML metamodel explicit. Not surprisingly, this shortage leads to ambiguities in practical usage of associations reported by experts [6].

The comparison also reveals two more flaws in the UML metamodel. First is the absence of meta-association *context* for meta-class Property. In fact, it means that the fundamental notion of Property is not completely defined in

UML. We consider this as one of the most serious problem of the entire UML metamodel (see [2] on the value of the Property construct in semantics of OO visual modeling).

The second problem is less fundamental yet is important for practical modeling: the meta-association *qualifier* is improperly defined in the metamodel. Our formalization clearly shows that the target of this meta-association is the meta-class of Roles rather than that of Properties. This mistake in the metamodel can lead to mistakes in practical modeling with qualified associations. Unfortunately, space limitations do not allow us to demonstrate the issue with a few remarkable examples we have in our archive (see [1] for one of them).

## 5 Conclusion

We have built a formal framework where the complex notion of association can be disassembled into a few basic blocks. We then composed from these blocks a few constructs that can formally model different aspects of associations as described and used in the Spec. We have found that semantics of the association construct can be well uncovered in a few Semantics and Description sections of the Spec, and is presented there in a sufficiently consistent way. However, the part of this semantics formally captured in the UML metamodel is much poorer and *that is why* the latter is ambiguous.

Our formal model allowed us to explain a few known problems with associations and detect several omissions in the metamodel, which have been unnoticed so far (see sections 2.3 and 4.4). We have also proposed a few general suggestions on augmenting and restructuring the metamodel for association towards capturing their semantics in a precise and unambiguous way.

## References

- [1] Z. Diskin. Visualization vs. specification in diagrammatic notations: A case study with the UML. In *Diagrams'2002*, Springer LNAI#2317, pages 112–115, 2002.
- [2] Z. Diskin and B. Kadish. Variable set semantics for keyed generalized sketches: Formal semantics for object identity and abstract syntax for conceptual modeling. *Data & Knowledge Engineering*, 47:1–59, 2003.
- [3] G. Génova, J. Llorens, and P. Martínez. Semantics of the minimum multiplicity in ternary associations in UML. In *UML'2001, 4th Int. Conference*,
- [4] C. Gunter. *Semantics of programming languages*. MIT Pres, 1992.
- [5] Dragan Milicev. On the semantics of associations and association ends in UML. Submitted for publication.
- [6] Dragan Milicev, Bran Selic, and the Authors. Joint E-mail Discussion, Fall 2005.
- [7] Object Management Group, <http://www.uml.org>. *Unified Modeling Language: Superstructure. version 2.0. Formal/05-07-04*, 2005.
- [8] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual. Second Edition*. Addison-Wesley, 2004.
- [9] Bran Selic. Personal Communication, Fall 2005.
- [10] P. Stevens. On the interpretation of binary associations in the unified modeling language. *Software and Systems Modeling*, (1), 2002.

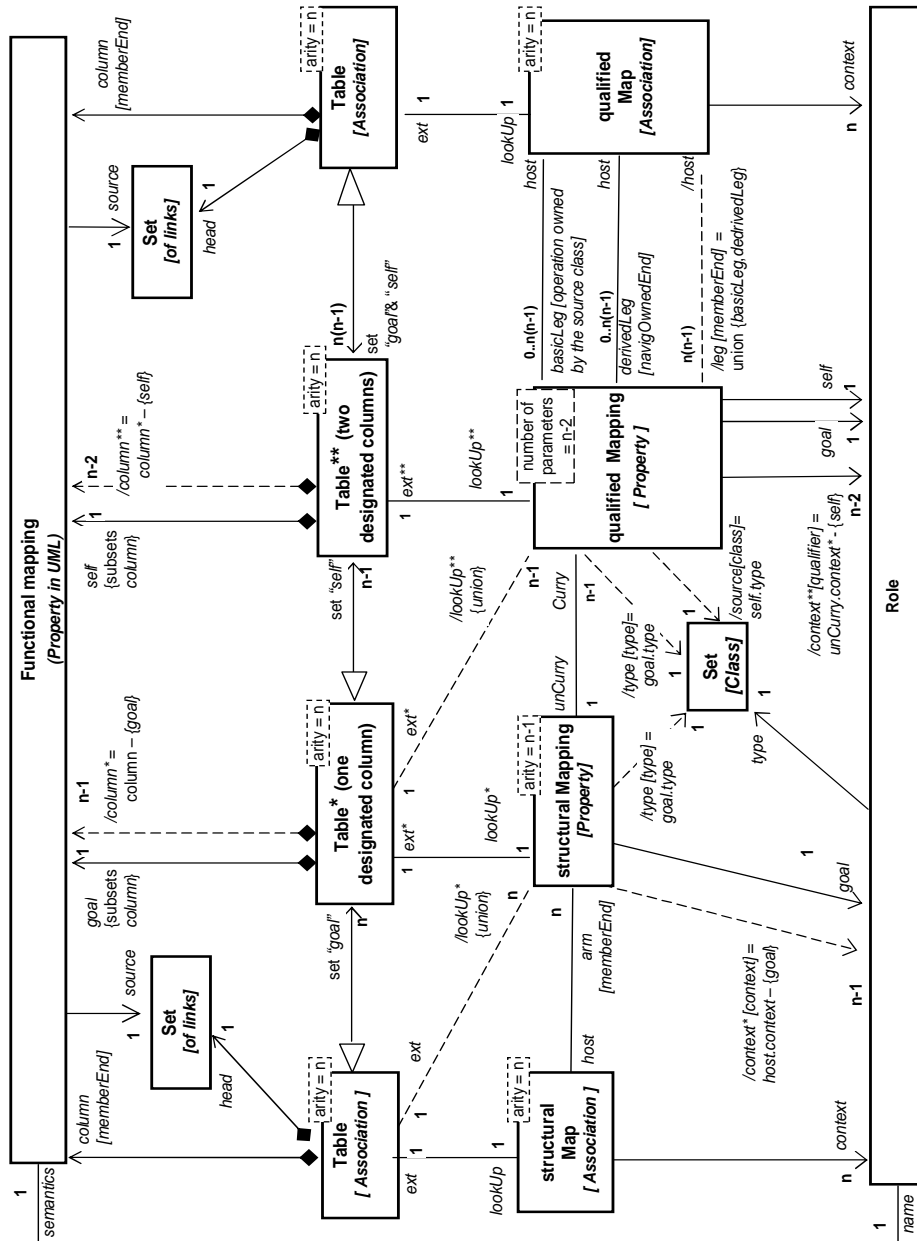


Fig. 3. Metamodel of our formal model for associations. Terms in square brackets refer to UML counterparts of our formal constructs. *Warning:* The node Table with its meta-associations is repeated twice to avoid clutter!