Visualization vs. specification in diagrammatic notations: A case study with the UML

Full version of 4pp. poster published in *Diagrammatic Representation and Inference. 2nd Int. Conf. on the theory and applications of diagrams, Diagrams'2002*. Eds. M. Hegarty, B. Meyer, N. Hari Narayanan. Springer LNAI No.2317, 2002, pp. 112-115

Zinovy Diskin*

Frame Inform Systems, Ltd., Riga, Latvia
Shulman & Kaufman, Inc., Detroit, USA
zdiskin@acm.org

Abstract. It is useful to see a notation as a visualization on the top of specification core. The latter deals with semantic meaning of notational constructs while the former is about its presentation (a user interface to specification). In the paper, this general paradigm is applied to a few major constructs used in the UML class diagrams: various sorts of associations between classes, including multi-ary and qualified.

A key peculiarity of the approach is that semantic meaning of notational constructs is also specified in a graphical (yet formal) language – the language of *generalized sketches* adapted from category theory. Then an UML class diagram appears as a *visual abbreviation* of semantic sketch hidden in it. By comparing two graphical images – the diagram and the sketch – one can analyze the UML notational mechanism and correct it towards better presentation of the intended semantic meaning.

1 Introduction

It was noted in [6] that

> discussions among computer scientists about notation can be more heated and protracted than discussions about issues usually considered more substantive. This suggests there may be more to notation than is admitted in phrases like "mere" syntax and "syntactic sugar".

And indeed, good notation is an invaluable means to communicate and discuss a problem and, moreover, to reason about the problem. Correspondingly, poor notation blocks these possibilities and enforces one to struggle with peculiarities of the notation rather then the problem. And even worse, poor notation can damage the conceptual framework used in the problem area and thus mislead one into a "linguistic swamp" of pseudo-problems caused by use of improper language: they at once disappear with a proper formulation (an illustrating example will be demonstrated in the paper).

Software engineering is really saturated with diagrammatic notations: they are invented, grow and expanded, mature (and begin to struggle between themselves). Nevertheless, despite a lot of attention, notation design is still an art rather than technology and it seems that making it more technological would be helpful. However, it is a highly non-trivial issue, as non-trivial as any other attempt to bring technology in an area very much determined by psychological and sociological factors.

* Supported by Grants 93.315 and 96.0316 from the Latvia Council of Science

So, the first step on this way might be very modest: to collect a certain amount of notational samples approved and used by experts, and try to view them in some systematic way, at least try to objectify why a notational construct should be considered good or bad. To do that, we need some scale, or template, against which notational samples might be evaluated and compared. In fact, we need a system of templates in correspondence with a system of factors – technical, psychological, social, cultural, which any complex notation features. Among these templates, a major one should be a semantic template on which notational constructs are to be evaluated w.r.t. how well they present the intended semantic meaning. After all, a minimal must of a notational construct is to present – denote! – some intended meaning well.

So, let D be a diagrammatic construct, $\mathcal{M}(\mathsf{D})$ its intended semantic meaning and the question is how to evaluate the quality of D as a presentation of $\mathcal{M}(\mathsf{D})$. A major problem here is that $\mathcal{M}(\mathsf{D})$ is often defined in informal and far non-elementary terms making decomposition and analysis of $\mathcal{M}(\mathsf{D})$, and correspondingly D, a very difficult task. In other words, $\mathcal{M}(\mathsf{D})$ and D appear as holistic entities difficult to analyze and reason, and D serves as a sort of illustrated $\mathcal{M}(\mathsf{D})$'s definition rather than $\mathcal{M}(\mathsf{D})$'s representation.

A treatment for the problem could consist in building a formal semantics for D, that is, describing $\mathcal{M}(\mathsf{D})$ in abstract terms of some formal language and thus viewing $\mathcal{M}(\mathsf{D})$ as composed from elementary blocks offered by the language. So, $\mathcal{M}(\mathsf{D})$ is described by some formal specification S having a certain structure against which D can be evaluated as a representation of this structure. For example, we can evaluate the quality of D w.r.t. of how much of that structure is preserved in D, or how adequately it is represented, or how suggestive is D for recovering specification S hidden in it. Thus, S appears as a *semantic invariant* of $\mathcal{M}(\mathsf{D})$ while D is its *changeable user-oriented representation*, a user interface to S.

The idea itself is clear enough, and actually was tried not once in the literature. However, it turned out hardly effective for *diagrammatic* notational mechanisms. The point is that popular formal languages normally used for building formal semantics – first-order logic and the like, Z, OCL [8] and others – are string-based formalisms so that S is a set of linear logical formulas (strings). Then correlations between a diagrammatic visual representation D and the underlying specification S are so fancy that one can hardly extract much useful information from their comparison.

The situation principally changes when for formalizing $\mathcal{M}(\mathsf{D})$ we use the framework of arrow diagram logic (ArrDL) and specify $\mathcal{M}(\mathsf{D})$ in terms of abstract sets and mappings. Then the formal specification S is also graphical and can be arranged in a special graph-based format of *sketch*. The latter is, roughly, a directed graph some fragments of which are marked with predicate symbols taken from a predefined signature. (The term *sketch* and methodology of specifying structures by sketches are taken from categorical logic [1], see also [4] for a brief presentation). The result is that we have two diagrammatic specifications: the diagram D and the sketch S, which can be related logically and visually. Moreover, for a reasonable D we might even expect some graphical similarity between D and S[1] so that D appears as a sort of *visual abbreviation* of S. On this base we can start to analyze D and reason about its quality as a representation.

What was just described may sound too hypothetical, and it is indeed so without examples. The goals of the present paper are thus two-fold. On one hand, we will see how the general schema above works in a particular case of diagrammatic notations really used in practice, and take for this a few major UML's constructs as samples to analyze. On the other hand, analyzing the UML notational mechanism is important by itself. The UML

[1] at least, it was observed in practice of applying sketches to software engineering problems

is an industrial standard in active development[2] and concrete recommendations towards improving the UML's notational mechanism, which we will derive from its sketch analysis, are valuable as such.

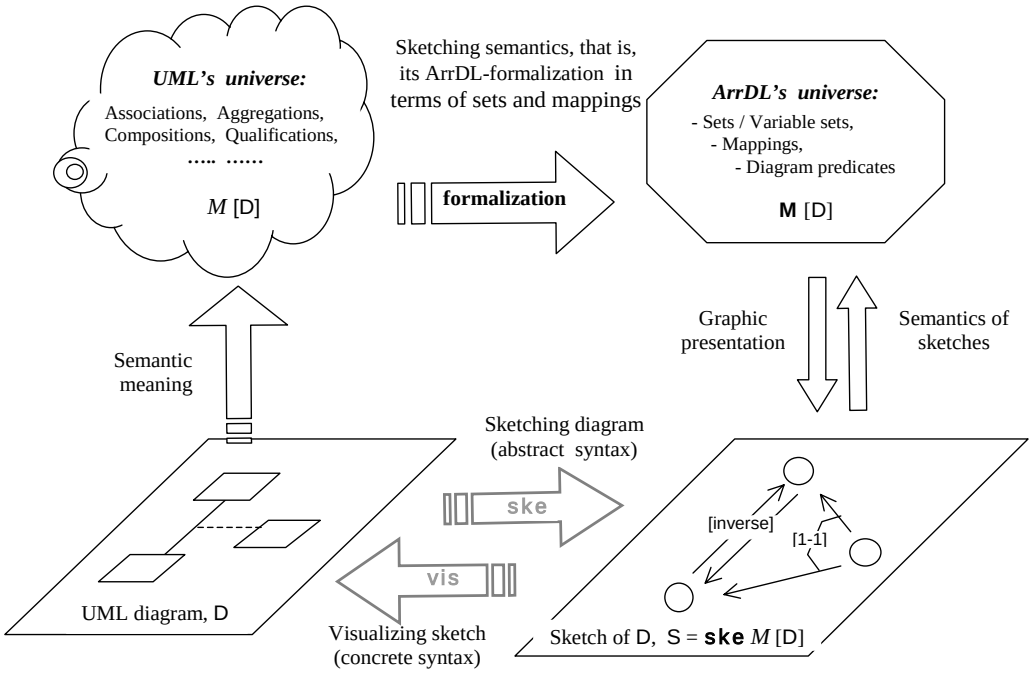


Fig. 1. Sketching UML-diagrams: general schema

The general schema of our work in the paper is presented on Fig. 1. We start with an UML diagrammatic construct D having a clear (yet informal) semantic meaning $\mathcal{M}(\mathsf{D})$. Then we formalize $\mathcal{M}(\mathsf{D})$ in terms of abstract sets and mappings – we will say *sketch* $\mathcal{M}(\mathsf{D})$, and specify our formal refinement M of $\mathcal{M}(\mathsf{D})$ by a graphical image, sketch S. Then we compare the diagram and the sketch and get some recommendations on improving notation towards better presenting the sketch, that is, the intended semantic meaning of D. Finally, we try to think out the lesson of D-S comparison in more general terms and formulate some general principles of a reasonable notational design.

Remark. A crucial question to the schema on Fig. 1 is whether the sketch language is sufficiently expressive to capture the meaning of the UML's constructs. The answer is affirmative (and, in view of extreme brevity of the sketch pattern, somewhat surprising): it was proven in CT that *any* formal construction can be specified by a sketch and thus, as soon as the meaning of an UML diagram is somehow formalized, it can be specified by a sketch as well. Of course, this principle possibility does not give any clue to how to formalize semantic meaning of a particular UML's construct and in each case some heuristic elaboration is needed.

The main work here was performed in [3], where an adequate formal semantics for each sort of association constructs used in the UML is built. It is shown in [3] that a really

[2] preparing the next version, UML 2.0, *qualitatively* enhancing the standard, is currently a hot issue for the community

adequate semantics for associations can be built only in the framework of variable sets, that is, sets whose extent changes with time. Nevertheless, in the present paper we will work in the world of ordinary sets and mappings – the static projection of the variable set world. It is sufficient for our goals of outlining a methodology of analyzing diagrammatic notations in general, and, on the other hand, does allow us to see a few important ideas on improving the UML's notation in particular.

The rest of the paper is organized as follows. In the next section we consider a very simple example of sketching an UML diagram and consequent notational analysis – just to show how our methodology can work. Section 3 is central, their a few types of association diagrams in the UML are examined along the lines we described. Two important consequences of the examination are (i) the solution of the infamous problem of multiplicities for N-ary associations and (ii) specifying constraints to qualified association, they are considered in sect. 4. Some attempt to summarize the results towards a more general view on the issue is outlined in conclusion.

2 Sketching UML diagrams: Getting started.

The notion of association between object classes appears to be a major construct in any reasonable language of conceptual modelling. As stated in [10],

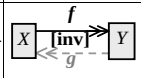
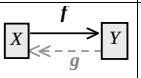
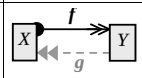
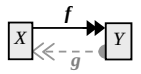
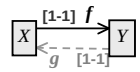
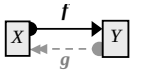
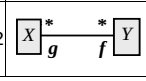
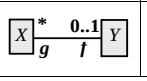
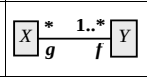
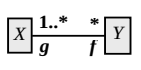
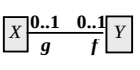
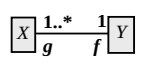
> Associations are the glue that ties a system together. Without associations, there are nothing but isolated classes that don't work together.

Similarly, mappings between sets form a key ingredient of the sketch view on the world. And even more, there is nothing in the world seeing through the sketch pattern apart sets and mappings. So, we begin with entering some notation and terminology for mappings.

2.1 Mappings between sets: notation and terminology

Given sets X,Y, we assume a general mapping between them $f\colon\ X \to Y$ to be partially-defined and multi-valued. The *domain* of f is the set $Dom(f) \subseteq X$ of all $x \in X$ for which $x.f$ is defined, and if $x \in Dom(f)$ then $x.f$ is a non-empty subset of Y. Here we write $x.f$ for the value of f at the argument x, we will also write $f(x)$ and sometimes $f.x$ for that. If f is *a priori* known to be single-valued then we consider $x.f$ an element of Y rather than a singleton subset of Y. The sets X and Y will be called the *source* and the *target* of f and denoted by $\Box f$ and $f\Box$ respectively. We call an arrow f *totally-defined* or just *total* if $Dom(f) = \Box f$.

Table 1. Properties of a general mapping $f\colon\ X \to Y$. Mapping $g\colon\ Y \to X$ is its inverse.

Semantics	f is partial and multi-valued 1	f is single-valued 2	f is total (hence, g is covering) 3	f is covering (hence, g is total) 4	f is injective (hence, g too) 5	f is a total single-valued cover 6
Sketch notation 1						
UML notation 2						

Dually to the notion of domain, the *range* of f, $Rg(f)$, is defined to be the set of all $y \in Y$ s.t. $y \in x.f$ for some $x \in Dom(f)$. Sometimes we will also write $f(X)$ for $Rg(f)$

and call it the *image* of X under f. We will call f a *cover* if $Rg(f) = f\Box$, this property is dual to being totally-defined.

A single-valued mapping is called *injective* or *1-1* if for any two different $x, x' \in Dom f$, $x.f$ and $x'.f$ are also different.

Finally, if we have two mappings $f\colon\ X \to Y$ and $g\colon\ Y \to Z$ such that $f\Box = \Box g$, then f and g can be composed into a mapping $(f;g)\colon\ X \to Z$ defined in an evident way: for any $x \in dom f$, $x.(f;g) \subset Z$ is defined to be the union of all sets $y.g$ for y running over $x.f \cap Dom(g)$.

To denote properties of mappings just described, we need some consistent notation, a suitable variant is presented in Table 1 in cells (1,1)...(1,5). The mapping g inverse to f is also shown, its properties are derived from f's properties. We will discuss this arrow notation in detail later.

2.2 Sketching a simple UML universe.

Consider the UML diagram D_1 in the 1st row of Table 2. Nodes denote object classes and the line segment between them denotes a binary associations; note also multiplicity constraints on the association ends (we use [10] as a standard reference to the UML: notation, terminology and explanations). Names on nodes and association ends make semantics of the diagram (in the static set framework) clear, and its abstract specification is presented on the right by sketch S_1. We have two abstract sets *Person* and *Company*, and two mutually inverse mappings between them. Since some multiplicity constraints are supposed to hold for the association as specified on the UML diagram, the mappings have special properties denoted by the corresponding special tails and heads explained in Table 1.

The fact that the two mappings constituting the association are mutually inverse is expressed by declaring the corresponding predicate [`inverse`] for them. Such a declaration is visualized by "hanging" the predicate label [`inv`] on the diagram of the two arrows (see [4] for precise meaning of "hanging" a marker onto a diagram). Note also that special arrow heads, tails and bodies used in the sketch are nothing but predicate markers/labels hung on the arrows. So, sketch S_1 consists of two nodes, two arrows and three diagram predicate declarations: [`total`](*employs*), [`cover`](*worksFor*) and [`inverse`](*employs*,*worksFor*).

2.3 Visualization vs. specification

Compare diagram D_1 and sketch S_1 in Table 2: the former appears as a special visual abbreviation of the sketch specification.

This abbreviation seems to be apt: it replaces three items (two arrows and the marker) by one item in a natural and clear-how-to-deabbreviate way, the more so that the UML allows (and encourages) naming association ends – these names are nothing but names of the arrows hidden in the association. Note, however, that if we consider composition of associations, then the absence of arrows could be a disadvantage.

An important aspect of the construct is how multiplicity constraints are denoted. The UML notation for them is clear and unambiguous, still a few (somewhat speculative) points could be discussed.[3] Note, first of all, that such properties of mappings / association ends as being total or covering are qualitative while in the UML they are denoted by numbers. A result is that the user should make (a quite trivial yet) computation to recover

[3] It seems that some flavor of speculation is difficult to avoid in discussing notation: too much in the issue is determined by too subjective factors like one's habits, preferences, cultural background *etc.*

the meaning and so a great advantage of diagrammatic notations – their gestalt-like perception – is weakened. It seems reasonable to denote mappings' qualitative properties somehow qualitatively, for example, like it is done in our notation introduced on Table 1.

An additional benefit of our visualization is that dual arrow properties of being total and covering are visualized somewhat dually too. If we enter some hypothetic duality predicate $[><]$, we may write this property of our visualization quasi-formally:

$$P_1 \overset{\mathcal{S}}{[><]} P_2 \text{ implies } (\mathbf{vis}P_1) \overset{\mathcal{V}}{[><]} (\mathbf{vis}P_2),$$

where P_i, $i = 1, 2$, denote the predicates in question and $\overset{\mathcal{S}}{[><]}$, $\overset{\mathcal{V}}{[><]}$ are realizations of the duality predicate in spaces $\mathcal{S}$ and $\mathcal{V}$ of, respectively, specification and visualization items. Further we will omit superscripts $\mathcal{S}$, $\mathcal{V}$ in similar formulas.

Also, with our choice of visualization means, a combination of arrow properties taken form the set {total, cover, single-valued} is visualized by the corresponding combination of visualizations (example is in cell (1,5) of Table 1). So, a useful principle of *visual superposition* holds: visualization of predicate superposition $P_1 + \ldots + P_n$ is superposition of the corresponding visualizations:

$$\mathbf{vis}(P_1 + \ldots + P_n) = \mathbf{vis}P_1 + \ldots + \mathbf{vis}P_n.$$

3 The Construct of Key: A Key Construct for the UML Class Diagrams.

Though less frequently than binary associations, multi-ary associations between object classes often appear in visual models. The importance of the construct was emphasized and explicitly designated so early as in the basic entity-relationship model [2] under the name of relationship. Since that a lot was written about multi-ary associations but still their proper general treatment – via the arrow diagram predicate of *separating* family of mappings or a *key* (see below) – is not known to the community.

3.1 General Format ....

Let's consider a family of single-valued mappings, $f_i\colon X \longrightarrow Y_i$, $(i = 1, \ldots, n)$ with a common source set X. We will call such a configuration *(n-)span* with X the *source* of the span and f_i its *legs*. Sometime we will refer to a span by calling only its source if the legs are clear from the context.

A span is called *separating*, or a *key* (to set X), or else *[1-1]-span*, if all the mappings f_i are total and separate the elements of X in the following sense:

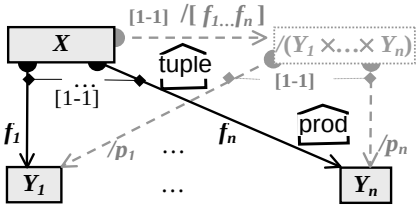
[1-1] for any $x, x' \in X$, $x \neq x'$ implies $x.f_i \neq x'.f_i$ for some i.

Any span (not necessarily separating) determines a tuple-mapping

$$f = [f_1 \ldots f_n]\colon X \longrightarrow Y_1 \times \cdots \times Y_n,$$

that sends an element $x \in \bigcap Dom f_i$ to the tuple $[x.f_1, \ldots, x.f_n] \in Y_1 \times \cdots \times Y_n$. Now it is easy to see that a family $(f_1, \ldots, f_n)$ is separating iff the mapping $f = [f_1 \ldots f_n]$ is total and one-one. This explains notation [1-1] for the condition above.

What we have just described can be graphically specified in the sketch language with the machinery of *diagram operations*, as shown on Fig. 2. Briefly, a diagram operation is an operation whose input and output are diagrams (configurations) of nodes and arrows. To distinguish basic items form derived, the former are shown with a filler and solid lines while the latter are not filled and drawn with dashed lines.[4] In addition, names of derived items are typed by grey color and prefixed with slash (this latter convention is borrowed from the UML).



Marker **[1-1]** hung on a span means that the span is separating. Marker **[prod>** denotes the *diagram operation* of taking Cartesian product: its input consist of a set of nodes (the sets Y_i in our case) and output is a [1-1]-span whose legs go into the input sets.

The characteristic property of Cartesian products is specified by another diagram operation, **[tuple>.** Its input consists of a Cartesian span, ($Y_1 \times ... \times Y_n$; $/p_1,...,/p_n$) in our case, and another span (X; $f_1,...,f_n$), whose legs have the same targets. The output is an arrow from X into the source of Cartesian span. In addition, if the span X is [1-1], then the arrow derived by the operation **[tuple>** is one-one.

Fig. 2. Key spans and their characteristic property.

So, if a set X is equipped with a key (separating family), its elements are uniquely identified by tuples and thus, X may be considered as a relation over sets $Y_1, ..., Y_n$ up to isomorphism. Conversely, if $R \subset Y_1, \ldots, Y_n$ is a relation, then the family of its projection mappings is separating. Thus, in a sense, the notions of relation and set with a key are equivalent.

3.2 . . . and its applications.

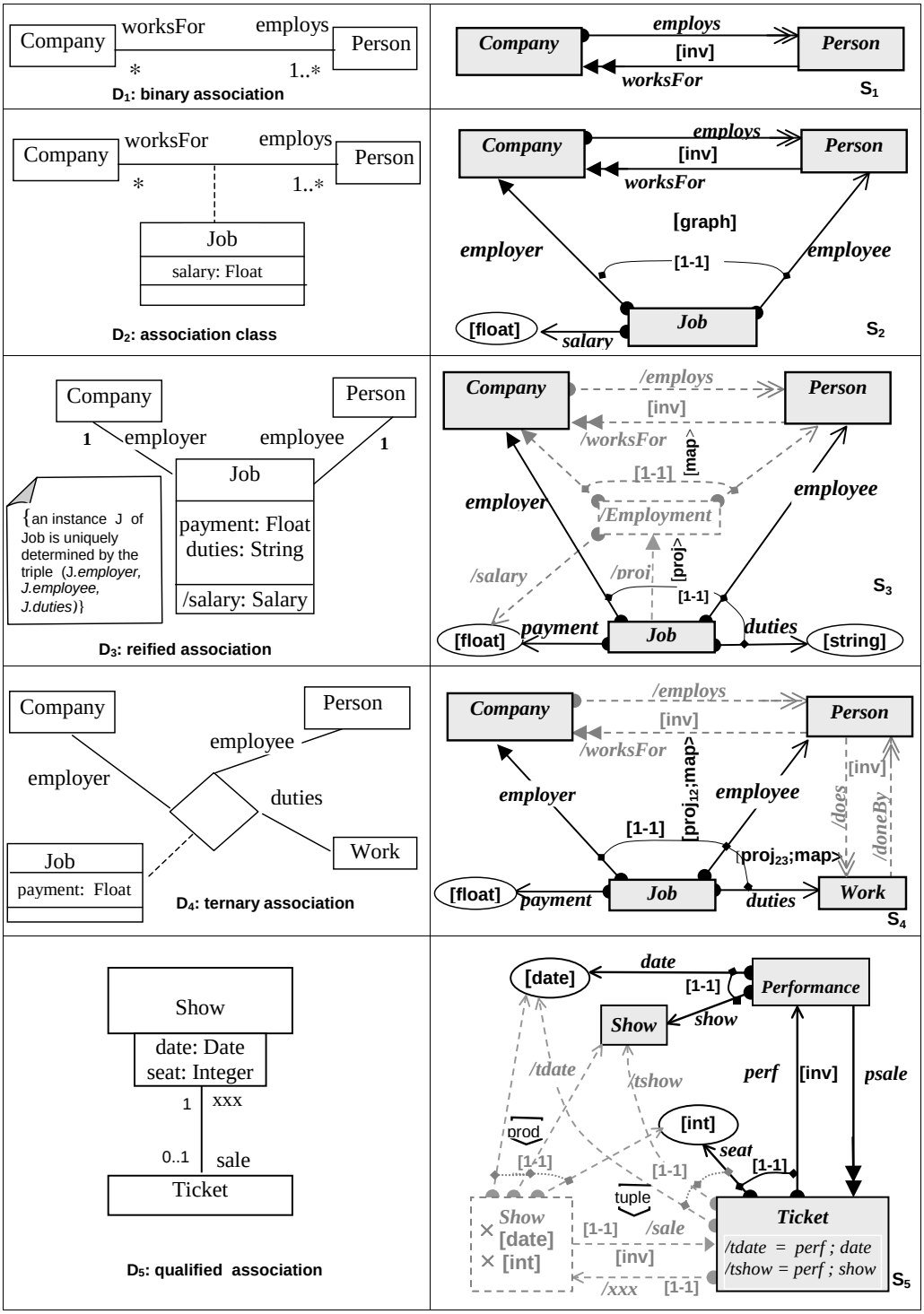
In the left column of Table 2 there are examples of simple UML-diagrams. Their meaning is clear yet informal. In the right column are precise sketch specifications where the key-predicate plays a really key role. We will consequently discuss each row of the table.

3.2.1 Association classes (row 2)**.** As we have seen in sect. 2, an UML's association is nothing but a pair of mutually inverse mappings (row 1). These mappings have a common graph that consists of pairs of objects linked by the mappings. Given an association, one can build its graph, and even make it a class – the class of association – but anyway the graph will appear as a derived item in the model (diagram). In contrast, the diagram D_2 describes a situation when an association is considered a class with its own attributes and operations from the very beginning. It means, first of all, that the set of links determined by the association-as-a-pair-of-mappings is now of interest on its own, in other words, the graph of the mappings becomes a basic rather than derived item with its own attributes attached. This is exactly the contents of the sketch S_2: all its items are basic and subjected to the three (apart multiplicities) diagram predicate declarations: [inv], [1-1] and [graph].

As explained in sect. 3.1, the marker [1-1] induces that the objects of the class *Job* can be identified with pairs (binary links) (C, P) with C a company and P a person. In

[4] The mnemonics behind "filled vs. blank" convention is as follows. In the database context, extent of a basic node has to be stored while extent of a derived node can be computed and thus does not need to be stored.

Table 2. Association diagrams as abbreviations of sketches



other words, the set *Job* is a binary relation over sets *Company* and *Person*. The marker [graph] states that this relation is the graph of (either of) the mappings: *employs* and *worksFor*.

3.2.2 Value domains. Note in sketch S_2 the oval node labelled by [float]. This label is not the name of the node but a predicate marker hung on it (i.e., on the diagram consisting of the node only). This marker constrains the extent of any node on which it is hung to be the same predefined set – value domain – of Float-numbers (and hence the name of the node becomes non-interesting and can be suppressed). In fact, we can consider value domains as 0-ary diagram operations producing nodes of known extent from empty input diagrams.[5] Attributes of a class are nothing but its mappings into value domains.

3.2.3 Reified and multi-ary associations (rows 3 and 4**).** Diagram D_3 presents a construct that is called *reified association* in the UML ([10]). An adequate semantics to the construct can be given only in the variable set framework (see Remark at the end of introduction) . As for its projection into the static sets world (our semantic universe for the present paper), reified associations differ from ordinary association classes by that they allow occurring the same link more than once into the class, that is, are multi-relations (bags, cf. [10, p.159]). For example, if a *Job*-object J is uniquely determined by the triple $(J.employer, J.employee, J.duties)$, then the same pair $(J.employer, J.employee)$ may appear more than once in the class *Job*.

The corresponding abstract semantic specification is presented by sketch S_3 where all that we need is done by the marker [1-1] hung on the 3-span as shown. Then set *Job* appear as a ternary relation over $Company \times Person \times$ [string]. In addition, one can extract from this ternary relation its binary projection on $Company \times Person$ (note the marker [proj⟩ of the corresponding diagram operation), and then present it by a pair of mutually inverse mappings for which the relation is the graph (note the diagram operation marker [map⟩). One can also derive an attribute */salary* of class *Employment*: for each of its objects, E, the value $E./salary$ is computed by adding payments for all jobs merged into E.

Sketching diagram D_4 is clear. A new object class *Work* appears on the stage and the mapping *duties* becomes a reference rather than an attribute. The triple $(employer, employee, duties)$ remains to be a key to the class *Job* so that the latter again can be considered as a ternary relation. In addition, two derived binary associations are presented in the sketch, they can be computed as it was explained above.

3.2.4 Qualified associations (row 5**).** The UML diagram in the bottom row of the table is taken from the book [10, p.48]. The association between classes *Show* and *Ticket* is considered *qualified* because of the two attributes (*qualifiers*) framed in a smaller rectangle attached to the corresponding association end. The intended meaning of the construct is as follows. Any object from the class at the qualified end of am association, a *Show*-object S in our case, together with a list of qualifiers' values, say, d : *Date* and n : *Integer*, determines not more than one object at the other end of the association. That is, while the value of $S.sale$ is a set of *Ticket*-objects, expression $S.sale[d, n]$ refers to only one (if any) *Ticket*-object.

Sketch S_5 specifies semantics in more detail. As it's seen from the sketch specification, actually we have two qualifications specified by key-predicates: *Performance*-objects are identified by *Show*s qualified by *date*s and *Ticket*s are identified by *Performance*s qualified by *seat*s. By composing arrows as shown on the sketch (in the *Ticket*-box), we get two

[5] In mathematical logic such empty-ary operations are usually called *constants*. Thus, value domains are constants in a predefined signature of diagram predicates.

derived mappings going out of class *Ticket*. Evidently, the family (/t*show*,/t*date*,*seat*) is also separating – note the derived predicate marker [1-1] declared for the 3-span. Owing to that, mapping

$$/xxx = [/tshow, /tdate, seat]\colon\ Ticket \to Show \times [\texttt{date}] \times [\texttt{int}]$$

is one-one (sect. 3.1) and, hence, its inverse *sale* is also one-one but is partially defined. Now, comparing the diagram and the sketch we can see that the UML diagram actually presents a view to sketch S_5: it can be mapped into some fragment of S_5's augmentation with derived items.

3.3 Visualization vs. specification.

We will begin with an on-the-surface analysis of Table 2 and then, in sect. 3.3.4, will try to formulate some more general view.

3.3.1 Binary association class: lost arrows. Diagram D_2 is a more or less direct abbreviation of sketch S_2. It has however some flaw in that two essential specification items – projection arrows *employer* and *employee* – are suppressed and the user cannot refer to them (simply because the diagram has no their names). This gap can lead to specificational problems in more complex cases when, for example, an association class participates in other associations and some constraints are imposed on the entire configuration (unfortunately, space limitations does not allow us to consider an illustrating example).

3.3.2 Attributes: arrows vs. strings. Another problem with the UML notation for association classes is caused by specifying their attributes by listing them in the class node-box. It is also an abbreviation since, as we have seen, an attribute is actually a mapping from the class to a value domain. No doubts, it is a reasonable notational tip which presents a set of attributes in a clear and compact way, much more compact than drawing a bundle of arrows and their value-domain nodes. However, thinking semantically, an attribute is an arrow and it may happen that some diagram predicate involves association arrows and attribute arrows too, just what we have in the situation of sketch S_3. In such a case, explicit arrow presentation of the attributes involved becomes a must for a proper visualization. This consideration shows that the same specification item can be visualized differently (within the same notational system) in function of context in which the item is visualized.

3.3.3 Qualified association: lost configuration. The bottom row of Table 2 presents a far too strong abbreviation. Sketch S_5 contains three basic nodes and five arrows while the UML diagram D_5 has only two nodes and four names for mappings involved: node *Performance* with arrow *show* are missed in the diagram. It is difficult to discuss diagram D_5 as a visualization of sketch S_5 because of their essential graphical dissimilarity. In fact, as it was already said above, the diagram presents a view to the sketch and can be mapped into not the sketch itself but into its augmentation with derived items.

One might argue that despite our critique, we were able to recover an actual specification hidden in the diagram. However, it was possible only due to the fact that semantics of the situation was clear from the very beginning owing to the names of the items in the diagram. Problems of the UML's way of visualizing qualified associations become explicit when we consider possible constraints that might be added to qualifications. We will see that in sect. 4.2 below.

3.3.4 Dissimilar visualizations of similar specifications. Comparison of the left and right columns of Table 2 is extremely instructive. The right column is logically and visually homogeneous, in fact, it presents a few variations of the same theme – identifying objects by a [1-1]-family of mappings. In contrast, in the left column we have externally different modelling constructs with quite different visualizations. Especially striking in this sense is comparison of the 3rd and 4th rows. The only difference between sketches S_3 and S_4 is that the target of arrow *duties* is a value domain in S_3 and an object class in S_4. However, the visualizations D_3 and D_4 of these semantically very similar situations are quite different.

An analogous situation we have with rows 2 and 4. Semantic specifications are similar and differ only quantitatively: 2-ary association in S_2 vs. 3-ary in S_4. However, their visualizations are essentially different: diagram D_4 shows projection arrows of the relation while diagram D_2 does not present them, instead, mappings of the relation considered as a graph are shown. This visualization "twist" was so unfortunate that created a whole (in)famous pseudo-problem of multiplicities for multi-ary associations, we will specially consider it below in sect. 4.1.

Considerations above might be formulated in our quasi-formal language of **vis-ske** mappings as **vis**'s incompatibility with some hypothetic similarity relations on the notational constructs: $\mathsf{S}_3 \sim \mathsf{S}_4$ but $\mathbf{vis}\mathsf{S}_3 \not\sim \mathbf{vis}\mathsf{S}_4$ and $\mathsf{S}_2 \approx \mathsf{S}_4$ but $\mathbf{vis}\mathsf{S}_2 \not\approx \mathbf{vis}\mathsf{S}_4$. However, this material can be seen from a different perspective. One might argue that minor changes in specification, say, when passing from sketch S_3 to sketch S_4, are nevertheless essential since they change the context. Correspondingly, if the UML diagrams visualize not only specifications but also their context, the essential differences in them become more understandable and justified.

We may even go further and try to make the notion of context more technical. For example, the context of an item can mean occurrence of the item into some other specification construct, and that affects this item's visualization. For instance, an attribute of a class is specified and visualized by putting the corresponding string into the class' node frame-box *unless* it occurs into some diagram predicate. In the latter case, the attribute must be visualized by an arrow going out of the class' node. So, a specification appears to be not a flat set of specificational items, rather, it is a complex system of specificational substructures. (Actually, it is just a particular illustration of a major semiotic thesis that signs always appear in sign systems, see [7] for a discussion). A proper visualization mechanism should smoothly map this complex system into a similar system of visualization items.

4 Two characteristic examples of specification problems caused by improper visualization.

As we have seen above, a few UML's modeling constructs appear as just different visual presentations of essentially the same semantic construction – a [1-1]-family of mappings. However, it turned out that in some situations these visualizations obscure a quite clear semantic picture and generate serious problems in understanding of visualized specifications. We will consider two characteristic examples.

4.1 Multiplicities for n-ary associations: *Much ado about nothing.*

The problem in the title appeared on the stage right after Peter Chen entered ER-diagrams into the world of visual modeling, and since then until now is still discussed (sometimes in a rather heated way) in the literature of both theoretical and industrial orientations.

Briefly, each of the ends of an N-ary association (each of the edges of an N-ary relationship) is supposed to be attached with a pair of integers (m, n) (including * as a possible value) specifying the potential number of values at the end when the values at the other $n-1$ ends are fixed [9, p. 3-73]. This definition is compatible with binary multiplicity but the precise meaning of the constraint for $N \geq 3$ is not clear. More accurately, a few interpretations were proposed, each having its own pluses and minuses, and what is the right concept of the constraints is still a question debated in the literature.

The most complete survey of the problem can be found in [5], which refers to more than ten recent papers specially devoted to the problem and to eight basic monographs on conceptual modelling and OOA&D touching the problem. The authors of [5] carefully analyze the existing approaches and on this base try to solve the problem. However, as they themselves state in the conclusion, the problem still remains unsolved and present an essential flaw in the UML 1.3.

There is nothing surprising in that failure. As soon as one has precise sketch specification, the entire problem becomes at once understandable as a typical case of a "linguistic game" with ill-formed terms. In other words, the problem of multiplicities for N-ary associations is actually a pseudo-problem caused by using an unsuitable specification language rather than the subject matter as such.

Indeed, as soon as we specify an N-ary association by the corresponding arrow span (the top row in Table 3), it becomes clear that multiplicity properties of the association are nothing but multiplicity properties of its projection mappings. The latter are ordinary mappings for which multiplicities have their ordinary meaning, and multiplicities of the i-th projection are not anyhow related to multiplicities of the j-th. In addition, projection mappings are always totally defined and single-valued so that the only qualitatively important property that can be declared for a projection arrow is whether it is covering. The latter is indeed important: we have seen above that covering properties of projections determine multiplicity properties of the binary associations between component classes (see sketch S_4 in Table 2 and its derived components). So, it is a must to specify covering properties of projection arrows in visual models of N-ary associations. However, with the UML's way of specifying N-ary associations, one cannot express such properties (see, *eg*, 2nd row in Table 3).

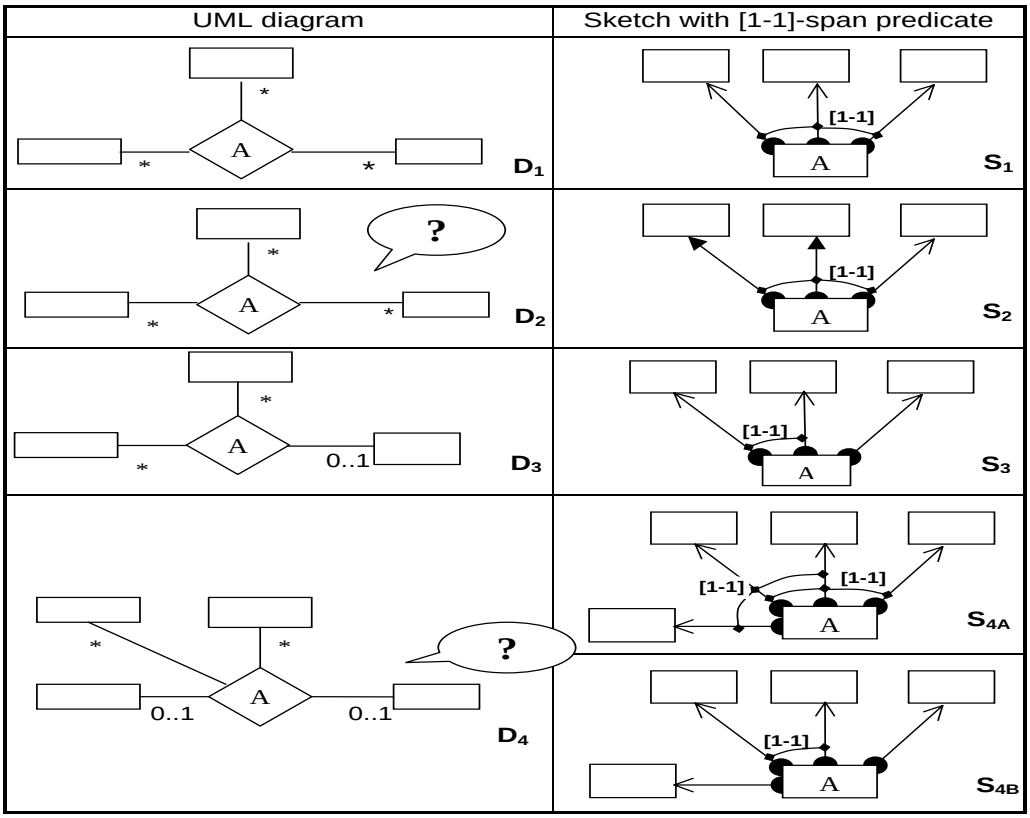
Another aspect of N-ary associations which is also included into their multiplicity properties actually have quite another nature. The point is that for a given N-ary association, that is, separating N-span $(p_1, \ldots, p_N)$, some combination of its projections, say, $p_{i_1}, \ldots, p_{i_k}$, $1 \leq i_1 \leq i_k \leq N$, can also have the separation property (in the relational data model terminology, the entire N-tuple of projections is then a superkey). For example, the actual meaning of UML-diagram D_3 in Table 3 is specified by sketch S_3 (compare it with sketch S_1). However, the conventional approach (inherited by the UML) to this type of constraints fails in a little bit more complicated situations. Consider, for example, sketches S_{4A} and S_{4B}: they specify essentially different associations[6] whose UML's representations coincide (diagram D_4). In other words, the UML's visualization cannot separate two different specifications and the mapping **vis** is not one-one.

4.2 Constraints for qualified associations: *As you like it.*

As we have seen in sect. 3.2.4, an UML qualification diagram can be considered as a very special visual abbreviation of sketch specification. The major problem with this abbreviation is that semantically meaningful node and arrow are hidden and thus the entire semantic picture is deformed. These hidden items may well have a substantial

[6] S_{4B} entails S_{4A} but not the converse

Table 3. Multiplicities for N-ary associations



semantic meaning rather than to play just an auxiliary role to explain what is qualification. Also, they may play an important role in specifying additional constraints to qualified associations, and without them such specifications become awkward – we will see that below in the section.

In the left column of Table 4 there are UML's qualification diagrams with various additional constraints (these examples, besides D_4, are taken from [10, pp.402-404]). The intended meaning of the constraints is explained in the text-boxes between the columns (also taken from [10]) . The precise semantics is specified by sketches in the right column.

As is well seen from the 1st row, the UML qualification diagram hides a semantically substantial node (class) *Entry* and this obscures the semantic picture. Indeed, consider the constraints C1 and C2. Their UML presentation involves a constraint *"same"* with unclear semantics; and probably this semantics will be different for diagrams D_1 and D_2. At the same time, sketches S_1 and S_2 clearly show that the constraints in question are nothing but multiplicity declarations for derived arrows obtained by arrow composition: $/fdir = entry; dir$ and $/fname = entry; name$. In the general situation, these mappings should be multivalued (as is shown on sketch S_0) since mapping *entry* is multivalued. The message of sketches S_1 and S_2 is just to state that, respectively, */fdir* is single-valued or */fname* is single-valued, that is, the constraint C1 or C2 holds. Also, sketch S_2 shows that the line segment *"same"* on diagram D_1 denotes, in fact, the operation of composing association links (one of which is now shown on the diagram !), which produces the right-hand-side association between nodes *Directory* and *File*. But even with sketch S_2 it is difficult to figure out semantics of *"same"* in diagram D_2.

Note, constraints C1 and C2 (as they are formulated textually) are very similar, and their similarity is explicated by the pair of sketch specifications $(\mathsf{S}_1, \mathsf{S}_2)$. In these sketches, we have the same constraint (single-valuedness of a mapping) declared for a derived arrow obtained in the same way by arrow composition as specified above. Not surprisingly that sketches S_1 and S_2 are geometrically similar. In contrast, diagrams D_1 and D_2 are both semantically and geometrically different. So, we again have the situation of $\mathsf{S}_1 \approx \mathsf{S}_2$ but $\mathbf{vis}\mathsf{S}_1 \not\approx \mathbf{vis}\mathsf{S}_2$.

The situation becomes even worse when we consider the constraints C3 and C4. For example, the intended meaning of C3 as it is explained in [10] is that a file may appear only once within a directory. This is precisely expressed in sketch S_3 which states that the pair $(E.dir, E.file)$ identifies an entry E in a unique way. Then (sect. 3.1), any pair $(D : Directory, F : File)$ either determines a unique entry $(D, F).[dir, file]^{-1} \in Entry$, or does not determine an entry at all. In the former case, the pair (D, F) determines also a unique name $(D, F).[dir, file]^{-1}.name \in [\mathtt{name}]$. Similarly, if a file may appear only once under some name but may have different names in different directories (or even the same directory), this is expressed by sketch S_4 symmetrically to sketch S_3.

As for the UML representation of these semantic situations, diagram D_3 is taken from [10] while D_4 is built specially for the present paper because the situation C4 is not considered in [10].[7] Thus, constraints C3 and C4 are similar but the corresponding UML diagrams are visually quite different. In addition, understanding semantics of diagram D_3 seems to be not an easy issue.

In general, it is difficult to avoid a feeling that textual descriptions of the constraints C1...C4 are much more clear than the corresponding UML diagrams, especially D_2 and D_3. It looks like that these diagrams have resulted from an attempt to put quite clear and simple semantic pictures into an unsuitable syntactical framework. Incidentally, [10, p.404] itself states that

> In practice, however, it is usually satisfactory to state the constraint textually, with the qualified association shown graphically.[8]

At the same time, the sketch specifications in the right column of Table 4 show that constraints to qualified associations can be quite naturally represented graphically, and in an elegant and transparent way. The problems with UML's representations are caused by the fact that the constraints in question are essentially *diagram* predicates. In contrast, their UML diagrammatic representations appear as a sort of *graphic interfaces* to *string-based* logical statements. Here we have a particular case of a general phenomenon: a graphic diagram visualizes a string-based statement. Such visualizations may be apt but often they are awkward or/and obscure semantics.

5 Conclusion.

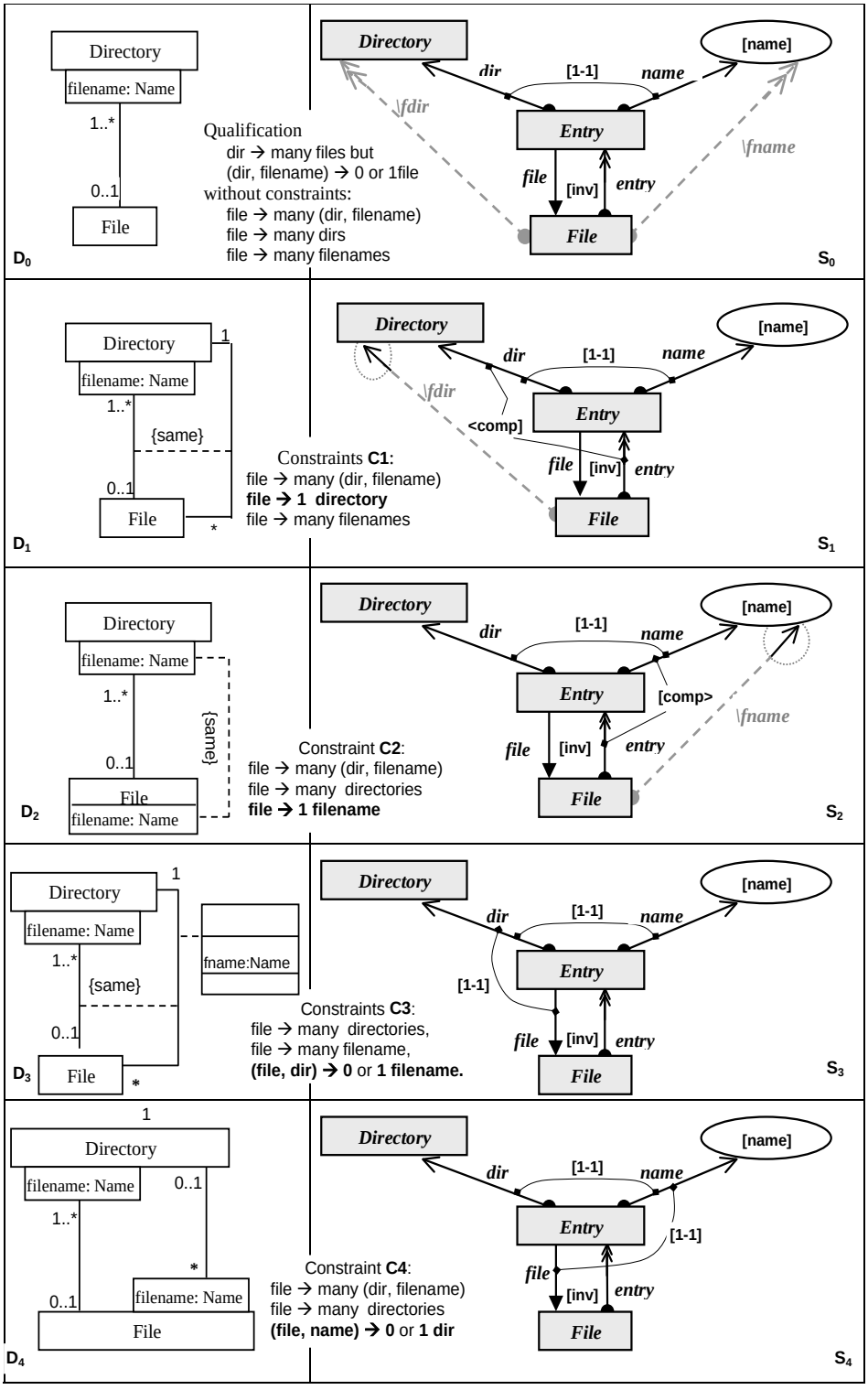
In the paper we have considered a few samples of examination of a pair D-S with D a basic UML's diagrammatic construct and S a sketch specifying (a formal refinement of) D's intended semantic meaning. We have observed a certain graphical similarity between D and S, which allowed us to consider and analyze D as a visual abbreviation of S.

The results of our analysis are two-fold. On one hand, we have obtained a few important recommendations on improving the UML's notation. The major one is that the

[7] It seems to be not accidental. Though constraint C4 is symmetric to C3, and it is well seen from comparison of sketches S_3 and S_4, this symmetry is obscured when one thinks of the situation syntactically (as the UML enforces) rather than semantically (as suggested by sketches).

[8] And, we'd add, the former is caused by that the latter is a poor notation.

Table 4. Constraints for qualification



UML will benefit greatly from adopting the basic notions of diagram predicate and diagram operation. In particular, the predicate of [1-1] arrow span is invaluable for specifying associations between classes.

On the other hand, we have tried to extract from our analysis some general notion of notational mechanism as a visualization mapping **vis**: $\mathcal{S} \to \mathcal{V}$ between spaces $\mathcal{S}$ and $\mathcal{V}$ of, respectively, specification and visualization items. Actually we have not specified $\mathcal{V}$ in detail, and have specified only $\mathcal{S}$'s components rather than $\mathcal{S}$'s organization (while our analysis has shown that $\mathcal{V}$ and $\mathcal{S}$ are hierarchically organized structures rather than flat sets of items). So, this side of our considerations was very fragmentary and approximate. However, we believe that it can be a starting point of future theoretical development.

Very likely that our results could be well interpreted and refined, and even formalized, in terms of Goguen's *algebraic semiotics*: spaces $\mathcal{S}$ and $\mathcal{V}$ are to be arranged as *sign systems* and **vis** as a *sign system morphism* [7]. At any rate, viewing a diagram D as a user interface to the underlying specification S – the viewpoint that algebraic semiotic would suggest – is in perfect match with what we did in the paper. Then certain theoretical foundations for the intentions we declared are already created and it remains only to apply them. Anyway, it's a theme of future research beyond the goals of the present paper.

References

1. M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall International Series in Computer Science, 1995.
2. P.P. Chen. The entity-relationship model – Towards a unified view of data. *ACM Trans.Database Syst.*, 1(1):9–36, 1976.
3. Z. Diskin and B. Kadish. Variable set semantics for keyed generalized sketches: Formal semantics for object identity and abstract syntax for conceptual modeling (55pp). To appear in *Data and Knowledge Engineering*.
4. Z. Diskin, B. Kadish, F. Piessens, and M. Johnson. Universal arrow foundations for visual modeling. In M. Anderson, P. Cheng, and V. Haarslev, editors, *Diagrams'2000: 1st Int. Conf. on the Theory and Applications of Diagrams*, volume 1889 of *Springer Lect.Notes in AI*, pages 345–360, 2000.
5. G. Génova, J. Llorens, and P. Martínez. Semantics of the minimum multiplicity in ternary associations in uml. In *"UML" 2001 – The Unified Modeling Language. Proc. 4th Int. Conference, Toronto, Canada, 2001*, volume 2185 of *Springer Lect.Notes in Comp.Sci.*, pages 329–341, 2001.
6. J. Goguen. On notation. In *TOOLS 10: Technology of Object-Oriented Languages and Systems*. Prentice-Hall, 1993.
7. J. Goguen. An introduction to algebraic semiotics, with applications to user interface design. In C. Nehaniv, editor, *Computation for Metaphors, Analogy and Agents*, pages II: 54–79. Univertsity of Aizu, 1998. The latest version is available at `http://ww-cse.ucsd.edu/users/goguen`.
8. IBM. *OCL Document Set*, 1997. Available from URL: `http://www-4.ibm.com/software/ad/library/standards/ocl.html`.
9. Object management group, OMG, OMG's web-page, `http://www.omg.org/library/schedule/Technology-Adopti` *UML Document Set*, 1997.
10. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.