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

In *Diagrams'2002: 2nd Int. Conference on the theory and application of diagrams*, Springer LNAI'2317, pp.112-115

Zinovy Diskin*

Frame Inform Systems, Ltd., Riga, Latvia
zdiskin@acm.org

In the world of diagrammatic notations, the Unified Modeling Language (UML) should be of special interest for cognitive studies. On one hand, UML integrates a host of different diagrammatic languages in real engineering use, and thus studying it should be itself extremely interesting. On the other hand, UML is adopted as a standard in software industry, and has already become a standard *de facto* in object-oriented analysis and design. This property provides a real practical value for UML studies. However, despite its dramatically increasing popularity, UML's drawbacks are well known and widely criticized. Currently, users and vendors associate their hopes on a better UML with the next version of the standard, UML 2.0, but it is commonly recognized that the problems standing in front of UML 2.0 are extremely hard [3]. So, careful cognitive analysis of UML appears to be an interesting, beneficial and urgent issue in the diagrammatic world.

However, before the instrumentary of cognitive science could be applied to UML, some key preliminary step must be done. On Peirce's semiotic scale ranging signs from icons to indexes to symbols, UML's notational constructs reside in the indexical-symbolic area, and some of them are close to the extreme symbolic end. Indeed, the meaning of many UML's constructs lives in the world of fairly abstract concepts. Hence, any reasonable study of the UML notation must begin with specifying precise semantics of the modeling constructs, otherwise their cognitive analysis will be built on sand. Unfortunately, the UML semantic volume and manuals provide rather vague description of constructs' semantics, and actually the latter has to be discovered and made precise, in ideal, formalized. This is a highly non-trivial issue, as any other formalization of substantial intuition underlying a complex domain, but our case has one more additional peculiarity.

Let's suppose that we have somehow fulfilled that hard work and formalized semantic meaning of a set of UML's constructs in, say, first order logic (FOL). Then we can recover a formal semantic meaning $\mathrm{M}(\mathsf{D})$ of any UML diagram D using these constructs, and specify it by a set of FOL-formulas (a theory), F. However, since F is just a set of strings, it would hardly help us in cognitive analysis of the diagram D. What we really need is a precise *diagrammatic* specification of $\mathrm{M}(\mathsf{D})$, say, by a diagram D_0, so that D_0 could serve as a sort of template against which the UML-diagram D could be examined and analyzed with the entire power of cognitive science.

Fortunately, a language for specifying complex constructions in a graphic yet formal way was developed in mathematical category theory, and then was adapted for software engineering needs (see [2] for a brief presentation). The underlying logic might be called *Arrow Diagram Logic* (ArrDL), and its specification format is called *sketch*. Briefly, a sketch is a directed multigraph, some fragments of which are marked with predicate labels taken form a predefined signature. The entire program outlined above could be called *sketching UML diagrams* and it should result in presenting UML-diagrams as

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

visual interfaces to sketches specifying their intended semantic meaning; Fig. 1 presents a general schema of the approach.

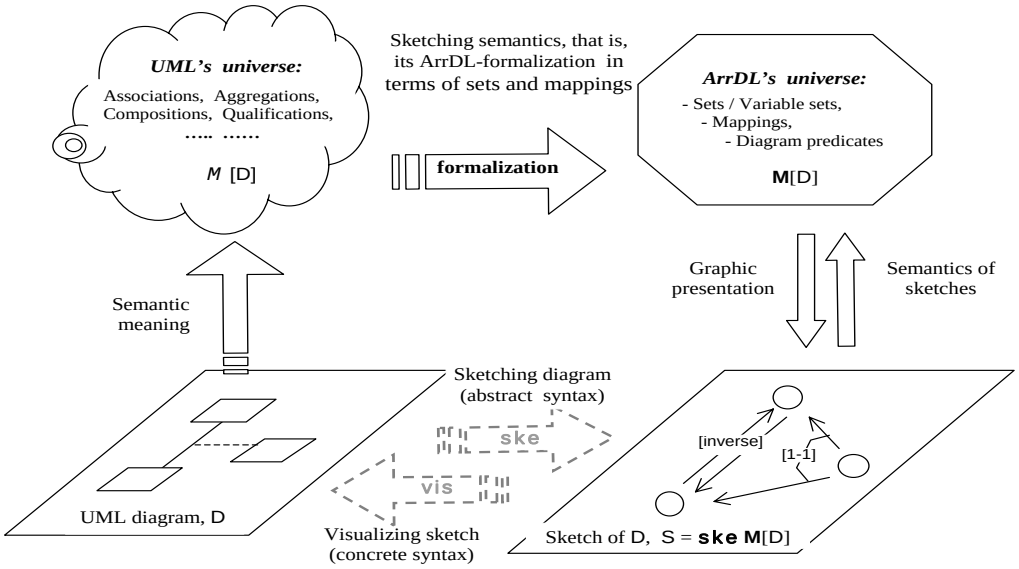


Fig.1. Sketching UML-diagrams: a general schema.

Having a UML-diagram D on one hand, and another diagram – sketch S – on the other hand, we can start a reasonable cognitive analysis of notational mechanisms used in D. Sketching a major UML's construct of association, including N-ary and qualified associations, was performed in [1]; some preliminary semantically-based analysis of UML notation was also outlined there.

A typical sample of the results is presented in Table 1. In the left column are UML's qualification diagrams: D_0 is a pure qualification and D_1 and D_3 contain possible additional constraints, their intended meaning is explained in the text-boxes between the columns. The precise semantics of the diagrams is specified by sketches in the right column. Nodes denote sets and arrows are mappings between them. Symbols `[1-1]`, `[inv]` and `[comp]` are predicate labels hung on their diagrams (indicated by thin auxiliary lines) and denoting diagrams' properties, `[comp]` actually denotes a diagram operation.[1] Figurative arrows' tails and heads are also predicate labels hung on trivial diagrams consisting of just single arrows. Semantics of the labels and other details can be found in the full paper [1].

Compare textual descriptions of the universe in the middle "column" with its UML (on the left) and sketch (on the right) specifications. It is difficult to avoid a feeling that textual descriptions are much more clear than the corresponding UML diagrams. It looks like that these diagrams have resulted from an attempt to put quite clear and simple semantic pictures into an unsuitable syntactical framework. In contrast, sketches make textual descriptions precise (even formal) and clear.

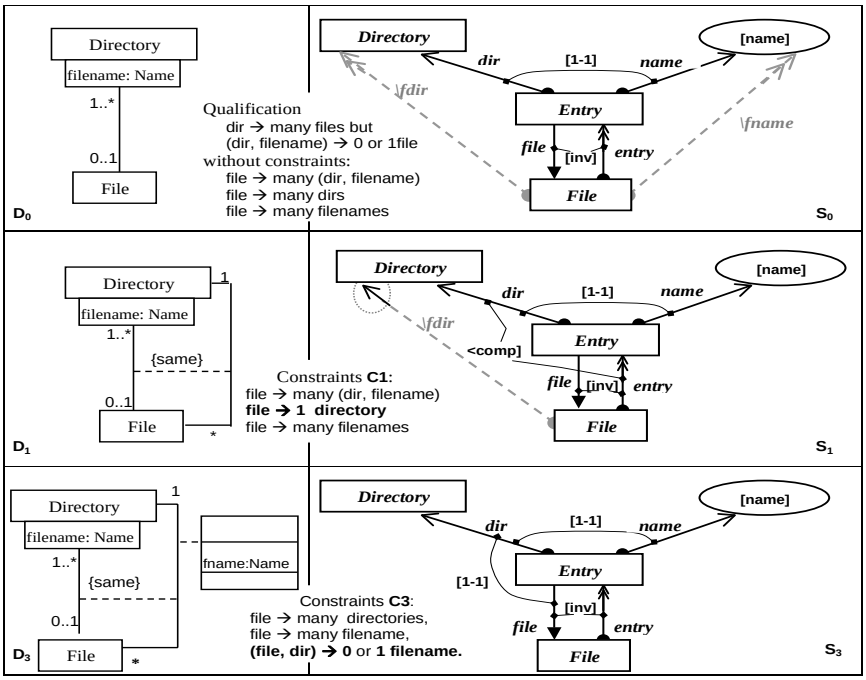
The situation with UML-diagrams becomes even worse when we consider constraints C2, symmetric to C1, and C4, symmetric to C3. Here the term *symmetric* means that the sketch specifications of these constraints, S_2 and S_4, are direct right-hand-side mirror images of sketches S_1 and S_3 resp. (hopefully, the reader can easily imagine them). In

[1] When one draws sketches on a computer display, explicating the diagram on which a label is hung is really easy: clicking the label highlights the diagram.

contrast to this nice graphic symmetry of sketches, the UML-diagrams specifying the constraints, D_2 and D_4 resp., are rather complicated and entirely non-symmetric to diagrams D_1 and D_3 – details can be found in [1].

The problems with UML's representations of the constraints in question are caused by the fact that these constraints are *diagram* predicates and should be correspondingly specified (see [2] for some details of ArrDL's syntax). In contrast, their UML diagrammatic representations appear as a sort of arbitrary *ad hoc* visualizations; such visualizations may be apt but often they are awkward or/and obscure semantics.

Table 1. Constraints for qualification



The full paper [1] presents other interesting examples of notational problems caused by that UML misses the fundamental role of the notion of diagram predicate. An important one is the infamous problem of multiplicities for N-ary associations widely discussed in the literature for a long time. It is shown in [1] that the multiplicity problem is actually a pseudo-problem induced by improper notation rather than the subject matter as such.

References

1. Z. Diskin. Visualization vs. specification in diagrammatic notations: A case study with the UML. Full version of the material in the present poster, `ftp://ftp.fis.lv/pub/diskin/Diagrams02.ps`, 2001
2. Z. Diskin, B. Kadish, F. Piessens, and M. Johnson. Universal arrow foundations for visual modeling. In M. Anderson *et al*, *Diagrams'2000: 1st Int. Conf. on the Theory and Applications of Diagrams*, *Springer LNAI*, volume 1889, 2000, pp.345–360.
3. C. Kobryn. Will UML 2.0 be agile or awkward? *Communications of the ACM*, 45(1):107–110, 2002.