On the Pragmatics of Model-Based Design

Position Statement

Reinhard von Hanxleden

Real-Time and Embedded Systems Group, Department of Computer Science
Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24118 Kiel, Germany
rvh@informatik.uni-kiel.de
www.informatik.uni-kiel.de/rtsys/

Linguists distinguish the syntax, semantics and pragmatics of languages. Together these three categories are referred to as semiotics—the study of how meaning is constructed and understood. All three categories can be applied to programming languages as well as natural languages. In the context of programming languages, syntax is determined by formal rules saying how to construct expressions of the language, semantics determines the meaning of syntactic constructs, and the pragmatics of a language refers to practical aspects of how constructs and features of a language may be used to achieve various objectives [1]. My position is that the pragmatics of modeling languages deserves more attention than it has received so far. Specifically, it appears that the practical issues of how to create, maintain, browse and visualize graphical models have been neglected in the past. This largely offsets the inherent advantages of visual languages, unduly limits designers’ productivity, and makes it difficult to design complex systems.

Traditionally, “pragmatics” refers to how elements of a language should be used, e.g., for what purposes an assignment statement should be used, or under what circumstances a level of hierarchy should be introduced in a model. It is usually not considered how the practical design activities themselves (editing, browsing, etc.) are performed—simply because this is usually not much of an issue when textual languages are concerned. There may be differences in convenience of use in different text editors, and integrated design environments (IDEs) can provide various levels of support in building and maintaining large software artifacts. However, the basic mechanics of writing or changing a line of code is rather standard and efficient. In comparison, the mechanics of editing a graphical model are much more involved, and it appears that there is much to be gained in this area. Hence “pragmatics of modeling languages” here refers to the practical aspects of handling a model in a model-based design flow.

There are several established fields that can provide valuable input here, such as the area of human computer interaction, cognitive psychology, and the graphical layout community. For example, there are fundamental practical differences in using textual or graphical languages [23], and freeing the modeler from the burden of manually drawing a graphical model opens the door to a number of productivity-enhancing techniques that allow to combine the best of both worlds [4]. Furthermore, there are already a number of paradigms well established in software engineering that could be put to use for model-based
design processes, including the design of the modeling infrastructure itself. For example, the state of the practice in creating a graphical model, say, a data-flow diagram or a Statechart, is to directly construct its visual representation with a WYSIWYG editor, and henceforth rely on this one representation. Instead one might apply the MVC paradigm \[5\] to separate a model from its representation (view). Together with a modeling environment (the controller/editor) capable of automatic model layout, one could thus provide flexible representations. These views could be adapted according to specific design activities, balancing useful information with cognitive complexity \[6\]. This would for example open the door for visual equivalents of folding editors \[7\], various forms of multimodeling \[8\], model hierarchies \[6\], and semantic focus-and-context representations \[9\] including its dynamic extensions \[10\].

References