The Importance of Ontological Metamodeling

Thomas Kühne

Darmstadt University of Technology, Darmstadt, Germany

1 Introduction

The terms “metamodel” and “metamodeling” are often used in an unqualified manner as if there were only one way to interpret them. Indeed, there is one common interpretation that applies very frequently—that of linguistic metamodeling [1]—that is, a metamodel is used as the language specification (i.e., grammar) for a level below in which the language is used. The so-called UML metamodel at level M2 in the OMG’s four-layer architecture is an example for a linguistic metamodel.

Another reasonable interpretation of “metamodel” is that of a medium specification, e.g., a repository format. The MOF, when used as a repository format for levels M1 and M2, i.e., not in its role as a meta-metalanguage at level M3, is an example for such a representation metamodel [2].

In this position statement, however, I would like to point out the utility of yet another form of metamodeling—ontological metamodeling [1]. In the following I argue that providing modelers with two levels only—a limitation which is imposed by many tools and notations (including the UML)—is creating problems that could be easily avoided.

2 Limitations of Two-Level Modeling

Fig. 1 shows a class diagram modeling an online store for computer hardware. As can be seen to the right of the dashed vertical line at the upper part of Fig. 1 the store needs to keep data about instances it sells (e.g., “m1”) but also their types (e.g., “mt1”). To the left of the vertical line at the upper part of Fig. 1 one sees that the modeler has applied the “Item Descriptor” pattern [3] in order to produce both user types and user instances at the level below. While the use of such (or similar [4]) patterns is common and nothing seems to be problematic about them at first sight, there are a number of disadvantages associated with such approaches:

- a new form of “instance-of” relationship is introduced (labeled “isOfType” in Fig. 1). This represents extra baggage on top of existing instantiation notions and has to be implemented manually.
- extra measures have to be taken in order to introduce typechecking among entities such as “m1” and “mt1” and/or inheritance between entities such as “mt1” and/or “ctl” and a common superclass such as “hardware part”.
- some types (here “Product” and its subtypes) do not represent any domain entities (→ in Fig. 1 they are not linked to black dots), but are only needed to setup the realization pattern.
Since designs such as the one in Fig. 1 not only try to capture the domain but also contain realization patterns it is often not clear what part of the structure is induced by the domain and what by the patterns. Furthermore, important information such as whether instances (e.g., “m1”) are allowed to access their types (e.g., “m1t”) for pieces of information cannot be specified. This is not surprising since objects (such as “m1t”) are not designed to control other objects.

The source of the problems associated with the design of Fig. 1 can be seen at one glance when looking at the essential distribution of modeling versus domain levels (which is only indicated through vertical dashed bars in Fig. 1). Fig. 2 illustrates that there are three domain levels to be modeled (bottom part) and that only two modeling levels (top part) are available to represent them. As a result some level compression has to occur; here both domain types and instances are represented using one modeling level.

### 3 Multiple Ontological Metalevels

The problems identifiable in the design of Fig. 1 can be easily addressed by providing as many ontological modeling levels as required. Fig. 3 shows the resulting design, if three ontological modeling levels are available. No more modeled “instance-of” relationship are required, all relationships can be supported by the language/infrastructure. Only elements representing domain entities (black dots at the bottom) exist. Simple modeling concepts may be used in order to specify visibility of type information for instances.
4 Conclusion

Supporting multiple ontological elements is rather straightforward [1, 2] and addresses the representation of multiple domain levels much better than currently used workaround techniques that must work within the limitations of a two-level modeling approach but cannot escape them. Accidental complexity introduced by realization patterns can thus be easily avoided and there is no ambiguity as to which part of the structure is induced by the domain or by realization patterns.

I therefore argue that ontological metamodeling is as important as the more widely known linguistic metamodeling and that notations such as the UML should be extended to support an arbitrary number, rather than just two, ontological user modeling levels.

References