Meta-level Independent Modelling

Colin Atkinson and Thomas Kühne
University of Kaiserslautern
67653 Kaiserslautern, Germany
{atkinson, kuehne}@informatik.uni-kl.de

ABSTRACT
The popularity of the UML and the recent focus on its extension mechanisms has raised the general awareness of the value and importance of metamodelling. However, the effectiveness of metamodelling with the UML is being hindered by the lack of suitable notational support. In particular the stereotype mechanism is not consistently applied to the UML's four-layer meta-architecture. The result is that different notations are used for the same concept at different meta-levels. In this paper we explain the motivation for a uniform, level-independent notation that supports the same concept in the same way regardless of its location in the meta-architecture, and then go on to suggest some of the principles upon which such a notation could be based.

1 INTRODUCTION

The widespread interest in the UML has increased awareness in the user community of the value and importance of metamodelling. Whereas previously metamodelling was viewed as the exclusive concern of tool builders and language designers, with the advent of the UML it has now entered the realm of every day modelling activities. In particular, tailoring (i.e., extending) the UML to one's special needs is expected to become a frequent activity. Even today, without support from a specialised profile mechanism, users can extend the predefined UML metamodel either indirectly by using stereotypes or directly by adding metaclasses.

Unfortunately, however, much of the potential power of metamodelling is currently lost because of the lack of a proper notation for working within a multi-level environment. Although the semantics of the UML assumes that users will generally need to influence at least three meta-levels (M0, M1 and M2), the UML notation assumes that they will only work at two (M0, M1). As a consequence, not only is the UML's notational support for metamodelling non-uniform and overly complex, but the semantics of the language is distorted and overloaded with superfluous concepts.

The reason for this imbalance is that until recently object-modelling notations were entirely focused on modelling at two levels. The third level only became relevant to users once they were given the opportunity to extend the modelling notation. Instead of further over-complicated extension mechanisms, what is required at this stage in the UML's evolution is the consolidation of the already available modelling mechanisms and the provision of a uniform, level-independent notation for their application.

In this paper we address the issues involved in meeting this challenge. The next section briefly describes the primary semantic and notational distortions that currently limit the effectiveness of the UML for metamodelling. The following section then goes on to suggest some notational conventions by which these problems could be rectified.

2 STEREOTYPING VERSUS INSTANTIATION

One of the most fundamental principles in object modelling is the instantiation[1] of a class to create an object. The UML notation for describing this situation is illustrated in Figure 1 below.

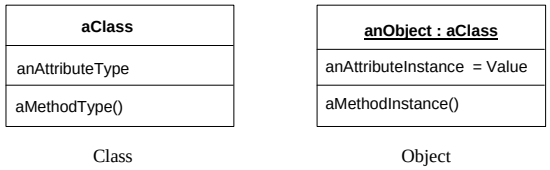

Figure 1: UML Instantiation Notation

[1] For the purposes of this paper we include the assignment of attribute values to attribute instances within the general term instantiation.

This notation has become the established way of modelling instantiation between the traditional model (M1) and data (M0) levels. Thus, `aClass` would reside at the M1 level and `anObject` at the M0 level. In the context of a multi-level model engineering paradigm it would seem fairly straightforward to generalise this notation to capture the instantiation concept between higher modelling levels, but unfortunately this is not the approach taken in the current version of the UML. Instead, the UML aims to provide a "shorthand" way of describing the instantiation of user-introduced modelling elements by introducing an additional set of semantic and notational concepts. The preferred UML representation of instantiation between the M2 and M1 level is illustrated in Figure 2.

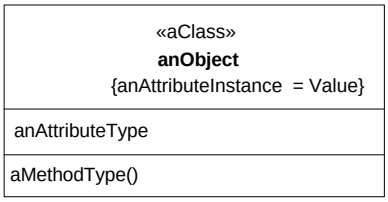

Figure 2: Stereotypes and Tagged Values

The motivation for this notation is to be able to describe instantiation (including the assignment of attribute values) without the need to model the class from which the instance is created, thereby avoiding direct extension of the M2 level. The class from which an instance is instantiated is represented by a "stereotype" (`aClass` in Figure 2), rather than by the ":" notation used at the M0 level. This class is meant to be a virtual metaclass from which `anObject` is instantiated [1]. The shorthand notation also allows attribute instances to be represented as tagged values (`anAttributeInstance` in Figure 2), but does not support the representation of method instances.

Although this method of representing instantiation may be "shorter" purely in terms of model volume (i.e. the amount of textual and graphical space needed to express the concept), it certainly is not "shorter" in terms of notational and semantic complexity. To understand instantiation at the M1 level, users have to understand the additional notational and semantic baggage associated with the stereotype concept. In effect, therefore, the stereotype mechanism represents an alternative way of expressing the instantiation relationship without offering any additional modelling power. The problem is amplified by the fact that the current UML instantiation concepts are not at all clearly explained and well understood. For instance, trying to pin down the exact properties of stereotypes and tagged values within a group of UML "experts", usually starts a long and controversial discussion.

The main problem, however, is not the nature of an individual instantiation-representation mechanism per se, but the fact that different mechanisms are used at *different levels*. Such redundancy would be justified if there were clear rules as to when one representation is applicable and why, but the variety of vaguely defined variants only creates confusion and hinders the application of a single mechanism uniformly across all levels. The fundamental requirement, therefore, is to remove the current ambiguities, inconsistencies and redundancies in the UML's approach to instantiation, and replace them with a uniform, level-independent approach.

3 LEVEL INDEPENDENT MODELLING

It is perfectly possible to model instantiation between arbitrary adjacent metalevels by extending the established notation for instantiation between the M1 and M0 levels. However, this can only be achieved in a clean way by appealing to two fundamental properties of a multi-level model-engineering environment.

3.1 Strict Metamodelling

The first important property relates to the definition of the levels themselves. The underlying assumption behind multi-level model engineering is that each level is an instance of the level above (except for the top level). Unfortunately, however, many modelling approaches, including the UML, apply this in a very loose way. They accept the general instance-of relationship between models, but do not apply it rigorously to all elements of the models. Thus, for example the UML metamodel mixes instances and the types from which they are created at the same level (e.g. class and object, association and link etc.)

Providing a clean notation is impossible when the concepts of metamodelling are not cleanly applied. Therefore, we assume a multi-level framework based on the doctrine of strict metamodelling [2] -

> ***Every*** *element of an M_n level model is an instance_of exactly one element of an M_{n+1} level model*

- with the understanding that some kind of "trick" is needed at the top level (e.g., entities at the top level are instances of other entities at the top level).

3.2 Clabjects

An importance consequence of metamodelling is that every instantiatable model element has both an instance facet and a type facet, both of which are equally valid. In other words every instantiatable

element is both a class (i.e. an instantiatable type) and an object (i.e. an instantiated instance). Non-instantiatable elements, such as those at the M0 level, by definition do not have the type facet.

One way of reconciling these two facets notationally is offered by the 3D visualisation of a clabject as a cube (see Figure 3).

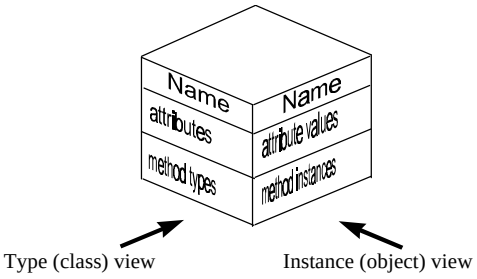

Figure 3: 3D visualisation of a clabject

The right hand face of this cube represents the instance (or object) facet of the model element, and contains the attribute instances and method instances derived from the type from which it was instantiated. The left-hand face represents the type view of the model element, and contains the attributes and method types which its instances will receive.

3.3 Uniform Instantiation Notation

The problem with the current UML notation is that it is not possible to represent both facets of a clabject in one symbol. By "flattening" this cube into two dimensions we obtain a representation of model elements capable of capturing both facets of an instantiatable element. The basic convention is that "instance" related features are indented with respect to the "type" related features to convey the idea that they are on the right hand face of the cube (see Figure 4).

This unified notation uses the well established instantiation symbol to indicate the type from which a clabject is instantiated. This is augmented by optional superscripts following the clabject names to identify their level. Assuming a strict metamodelling framework, the following identity must always hold between these superscripts, except at the top level.

$$X = Y - 1$$

The final notation suggestion, not illustrated in Figure 4, is to use the convention of underlining names to differentiate instantiatable clabjects from non-instantiatable objects, the latter being the entities underlined.

3.4 The Good, the Bad and the Ugly

With the availability of a uniform notation of the kind described in the previous section, two of the three current UML lightweight extension mechanisms would become superfluous. Constraints (the good) would still be extremely useful, since the application of additional restrictions to modelling concepts is still going to be valuable regardless of the manner in which they are defined. However, stereotypes (the bad), and tagged values (the ugly) would no longer add any expressive power beyond the regular meta-instantiation mechanism. Stereotypes would just provide an alternative (although more complicated) way of defining the type from which a model element has been instantiated, while tagged values would just provide an alternative means of representing attribute values. Using the clabject notation, the corresponding tags would simply have to be defined as attributes in the corresponding metaclass.

The adoption of a uniform mechanism of the form defined above would therefore have no negative consequences, since the effects of tagged values and stereotyping would be still available, but would lead to a significant improvement in the clarity and simplicity of the UML.

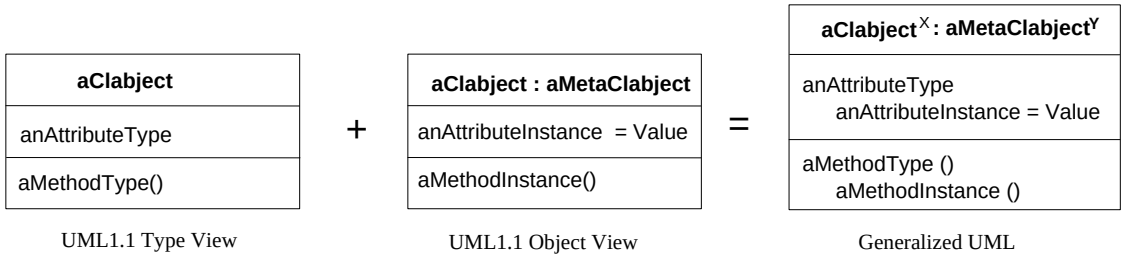

Figure 4: Uniform Instantiation Notation

4 CONCLUSION

The UML currently employs different concepts and notations to represent instantiation depending on the levels at which the instantiation takes places.

This non-uniform modelling approach not only complicates and distorts the notation and the semantics of the UML, but also limits the potential effectiveness of metamodelling.

In this paper we have described two fundamental principles upon which a clean, multi-level model engineering paradigm should be based:

- strict metamodelling and
- clabjects.

We then went on to suggest a way of enhancing the well-established instantiation notation to provide a uniform way of depicting the instantiation of a model element, independently of the level at which the instantiation takes place. A minor generalisation of this form will allow the existing "built-in" extension mechanisms (except constraints) to be removed from the UML, with a consequent overall simplification in its semantics and notation. Such a simplification is likely to be of particular value in connection with the expected shrinkage of the UML kernel [3], and the increased reliance on metamodelling-based tailoring mechanisms such as profiling [4].

The suggested notational enhancements are not the main point of the paper, however. The main message of the paper is that future versions of the UML must fully embrace the principle of multi-level model engineering by providing a uniform, level-independent set of modelling concepts and notations. Only then will the true benefits of model engineering become available to users of the UML.

5 REFERENCES

1. C. Kobryn (ed.), OMG Unified Modelling Language Specification, Version 1.3, OMG document ad/99-06-08 1999
2. C. Atkinson, Metamodelling for Distributed Object Environments, First International Enterprise Distributed Object Computing Workshop (EDOC'97). Brisbane, Australia, 1997
3. C. Kobryn, UML 2001: A Standardization Odyssey, Communications of the ACM (42):10, p. 29–37, 1999.
4. P. Desfray (ed.), White Paper on the profile mechanism, OMG Document ad/99-04-07, 1999