REGULAR PAPER

Cesar Gonzalez-Perez · Brian Henderson-Sellers

A powertype-based metamodelling framework

Received: 9 November 2004 / Accepted: 31 March 2005 / Published online: 14 November 2005
© Springer-Verlag 2005

Abstract  Software development methodologies may be described in the context of an underpinning metamodel, but the precise mechanisms that permit them to be defined in terms of their metamodels are usually difficult to explain and do not cover all needs. For example, it is difficult to devise a way that allows the definition of properties of the elements that compose the methodology and, at the same time, of the entities (such as work products) created when the methodology is applied. This article introduces a new approach to constructing metamodels and deriving methodologies from them based on the concept of powertype. It combines key advantages of other metamodelling approaches and allows the seamless integration of process, modelling and documentational aspects of methodologies. With this approach, both methodology components and project entities can be described directly by the same metamodel.

Keywords  Metamodelling · Powertype · Software development methodologies

1 Introduction

Models are in increasing use in software development. They are used to represent a system of some kind. For instance, a software developer may create a model to represent either the software system as it is (a representation) or as a blueprint for a system yet to be built (a specification) [23]. Here we concentrate on one particular kind of model – a metamodel, which can be defined as a model of a model [7].

In the object-oriented software development space, we can focus on metamodels as being models of a methodology, which itself encompasses both process aspects and product aspects, including modelling languages such as the UML (Unified Modelling Language [18]). Once such a metamodel has been created, it can be used to create new, or validate existing, modelling languages useful for software developers. The rules established by the metamodel can also be encapsulated in software tools such as CASE tools. In addition, standards such as UML and SPEM (Software Process Engineering Metamodel [19]) are defined by their metamodel (the so-called abstract syntax). Such metamodels must therefore capture in a standardized yet highly accessible fashion all the features of the modelling language itself.

Although much of the following discussion is generally applicable to both modelling languages like UML and to processes like those that can be generated from the SPEM or similar process metamodels such as the OPEN Process Framework metamodel [11], our focus here will be on the less well understood area of process metamodels. Both UML and SPEM are described not just as a pair of relative metalevels but as two out of a four level hierarchy, labelled M0 through to M3 (Fig. 1); where, according to this OMG-adopted architecture, the metamodel as described above occupies level M2 and software developers use elements within the M1 and M0 space. This means that the developers themselves should never need to have inside knowledge

![Fig. 1 The OMG four layer (a.k.a. four level) architecture, used originally for UML and MOF (after [13])](image-url)
A powertype-based metamodelling framework

Fig. 2 UML diagram for the defined methodological concepts (shown as packages on the left hand side). Each concept is represented as a class, with relationships between concepts as associations. Here, we use the UML white diamonds to denote *any* whole/part relationship – since the UML does not specify tight semantics for black and white diamonds (e.g. Barbier et al., 2003), the symbol has to be user-defined. In this diagram, the upper level classes are the most abstract.

of the M2 level. Consequently, everything that is needed in the M1 construction of models or processes (together these contribute to the full “methodology”\(^1\): Fig. 2) *must* be defined by the M2 level metamodel. For process particularly, this gives rise to some significant problems, the solution of which is the topic of this paper. The several facets of that problem are the following:

- Consider the definition of a coding task in a methodology. When enacted\(^2\) on a project (i.e. when an actual instance of the task is created at the M0 level), the coding task has, say, a duration of one week. This means that, for this enacted coding task to have a value for one of its attributes, that attribute must be defined (name and type) at the next higher metalevel (here M1). However, since all tasks have durations, it would seem more useful to reflect this statement (that all tasks have a duration attribute) at the level where the generic task class is defined. This is at level M2. In the OMG four layer architecture, the relationship between levels is stated to be that of “is-an-instance-of”. The problem is that if an attribute of duration is defined at the M2 level (as would be expected), then at the M1 level, the instantiation relationship requires it to have a value; whereas as we have just seen the software developer requires it to be defined at the M1 level and given a value at the M0 level. This is a paradox that we address in this paper.

- When working on a project, Mary creates a class diagram and delivers it to her manager, Peter. This is clearly part of the process enactment so, from a process perspective, this class diagram could be said to reside in the lowest metalevel (M0). However, Tom is a UML expert and when he follows the OMG’s UML metalevel hierarchy he notes that the classes depicted in the diagram are part of the M1 model that he would expect a software developer to develop. Here is a contradiction since Susan’s cannot be both M0 (process viewpoint) and M1 (modelling viewpoint) contemporaneously.

These can be summarized as problems of “generation” and of “enactment”.

Part of the problems illustrated above results from an apparent mismatch between intuitive modelling demands (for example, all tasks need durations at M0) and an adherence to the notions of strict metamodelling (as in all the OMG standards) in which instance-of relationships are the only permissible relationship between Mx levels and, at the same time, it is forbidden to have an instance-of relationship inside a single metalevel.

Finally, before we become embroiled in a technical discussion of the problems of multi-layer metamodelling, we need to ensure that an understandable and consistent terminology is being used. This is necessary since much of the literature uses disparate terms or terms with the same name but totally different semantics. Table 1 presents a set of definitions of technical terms as used in this paper in order to facilitate this common understanding of process/methodology focussed terms. These terms follow standard terminology whenever this exists.

In the next section we elaborate on these particular problems, outlined briefly above and then discuss some of the

---

\(^1\) Later, we will challenge the appropriateness of the M0 to M3 nomenclature; for the OMG-focussed discussion here, however, we retain them for consistency with the published OMG standards.

\(^2\) For definitions of technical terms see Table 1.