Information and Software Technology 55 (2013) 2099–2124

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Ontological and linguistic metamodelling revisited: A language use approach

Owen Eriksson [a,*], Brian Henderson-Sellers [b], Pär J. Ågerfalk [a]

[a] Department of Informatics and Media, Uppsala University, Box 513, 751 20 UPPSALA, Sweden
[b] School of Software, University of Technology, PO Box 123, Broadway NSW 2007, Sydney, Australia

ARTICLE INFO

Article history:
Received 4 July 2012
Received in revised form 31 May 2013
Accepted 19 July 2013
Available online 9 August 2013

Keywords:
Concepts
Speech act theory
Set theory
Metamodel

ABSTRACT

Context: Although metamodelling is generally accepted as important for our understanding of software and systems development, arguments about the validity and utility of ontological versus linguistic metamodelling continue.
Objective: The paper examines the traditional, metamodel-focused construction of modelling languages in the context of language use, and particularly speech act theory. These concepts are then applied to the problems introduced by the "Orthogonal Classification Architecture" that is often called the ontological/linguistic paradox. The aim of the paper is to show how it is possible to overcome these problems.
Method: The paper adopts a conceptual–analytical approach by revisiting the published arguments and developing an alternative metamodelling architecture based on language use.
Results: The analysis shows that when we apply a language use perspective of meaning to traditional modelling concepts, a number of incongruities and misconceptions in the traditional approaches are revealed – issues that are not evident in previous work based primarily on set theory. Clearly differentiating between the extensional and intensional aspects of class concepts (as sets) and also between objects (in the social world) and things (in the physical world) allows for a deeper understanding to be gained of the relationship between the ontological and linguistic views promulgated in the modelling world.
Conclusions: We propose that a viewpoint that integrates language use ideas into traditional modelling (and metamodelling) is vital, and stress that meaning is not inherent in the physical world; meaning, and thus socially valid objects, are constructed by use of language, which may or may not establish a one-to-one correspondence relationship between objects and physical things.

1. Introduction

Conceptual modelling is used in information systems and software development to enable conceptual understanding of the systems, the information they contain, and the processes by which they come about. Conceptual models facilitate the understanding of information systems requirements and, more generally, enhance the quality of information systems development. Conceptual modelling is a term commonly used to indicate modelling in the software context that is independent of the constraints of programming languages. It focuses on the description of real-world (business-focused) problems and how to represent them in models and could therefore be loosely related to requirements engineering, business systems analysis, and enterprise engineering.

Modelling uses a modelling language to communicate information about the models, be they for system design or for processes and methodologies. A modelling language consists of, *inter alia*, an abstract syntax, a concrete syntax (notation)[1] and semantics. To ensure quality and consistency, modelling languages need to be clearly defined so that their use is consistent across development teams, countries, etc. Much of the work in modelling languages for computing contexts (software engineering, information systems) over the last several decades has been focused on 'general purpose modelling languages', such as the now standard Unified Modeling Language™, UML® [1,2]. More recently, domain-specific modelling languages (DSMLs) have been intensively researched and developed to practical solutions [3]. Here, we focus on general purpose modelling languages and eschew discussions about DSMLs.

Although there are several ways of writing down formal definitions of modelling languages, one frequently used is that of the metamodel, defined as 'a model of models', which defines the abstract syntax, often itself expressed using UML's notation (typically a class diagram) together with additional behavioural and semantic constraints (perhaps using OCL or a similar logic-based

* Corresponding author. Address: Department of Informatics and Media, Uppsala University, Box 513, 751 20 Uppsala, Sweden.
E-mail address: owen.eriksson@im.uu.se (O. Eriksson).

[1] Although an important element, we do not discuss the notational aspects of a modelling language in any detail in this paper.

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.07.008

language). Metamodels focus on the concepts and rules of the methods and models used in information systems development and software engineering [4]. Metamodelling, i.e. how to create metamodels, is an important research area because there is a need for shared conceptualisations in software and systems engineering in order to avoid misunderstandings and barriers of communication in the development of IT systems [5]. Consequently, the application of metamodels is increasingly recommended. The use of metamodels for conceptual modelling, especially in the context of software engineering, has a relatively short history, in comparison to its much longer use in database modelling. The (early, 1991) CDIF standard was based on a three-level hierarchy (as summarised more recently in [6]). A few years later, suggestions by Carmichael [7] and Henderson-Sellers [8] led to the proposal to use this approach to bring together the plethora of object-oriented modelling notations then extant [9,10]. The first standard to utilise this idea of metamodelling for object technology standards was the UML [1], standardised by the Object Management Group (OMG), which introduced the four-level hierarchy of Fig. 1, with an expressed aim of providing an infrastructure "to support the creation, manipulation and interchange of meta models" [11]. It provides a metametamodel at the top layer, called the M3 layer (or MOF: Meta-Object Facility). An M3-model defines the language that is used for building metamodels, which are understood to define modelling languages at the M2 layer. The most prominent example of a Layer 2 metamodel is the model that defines the UML. These M2-models describe elements of the M1-layer, which are thus models written in UML. The elements in these M1 models each represent concepts pertinent to the real-world (or computer world) domain under investigation. Individuals classified as conforming to a concept are often called 'instances'. An instance is seen to be atomic, as compared to concepts and classes that can be represented mathematically by sets or categories e.g. [12]. (Other multilevel constructions are well presented in [13], wherein the linking of language-based and model-based approaches is similar to that in [14].) However, in the end (p. 290), the author [13] eschews a language use focus in favour of the (to-date) traditional structural (a.k.a. linguistic) metamodelling framework, as employed by the OMG.

"The primary responsibility of the metamodel (M2) layer is to define a language for specifying models" whereas the purpose of the M1 layer is said to be to define a language that describes an information domain [1, pp. 7–8]. Thus, Class belongs to the M2 layer and Horse (an instance of Class) is part of the M1 domain language. The last layer is the M0-layer, said to describe run-time instances [15, p. 17] or objects of the model – thus emphasising UML's focus on software objects rather than objects in the real world. Notwithstanding, some authors interpret 'M0' as being the 'real world' – as seen in some diagrams abstracted from extant literature.

In the first versions of UML, there was an M2 class called Object, which led to confusion as to whether instances of this class existed at level M1 or at the more natural M0 [16]. Consequently, in Version 2, the M2 Object class was replaced by InstanceSpecification. This allows for instances of InstanceSpecifiation to be created as part of the M1 model, despite referring to an M1 entity. The UML documentation [15] thus notes that the instances at the M0 level should not be confused with an instance of an InstanceSpecification, which is used simply as an illustration (a snapshot) of the class, e.g. Prancer:Horse (or the more anonymous:Horse) at the

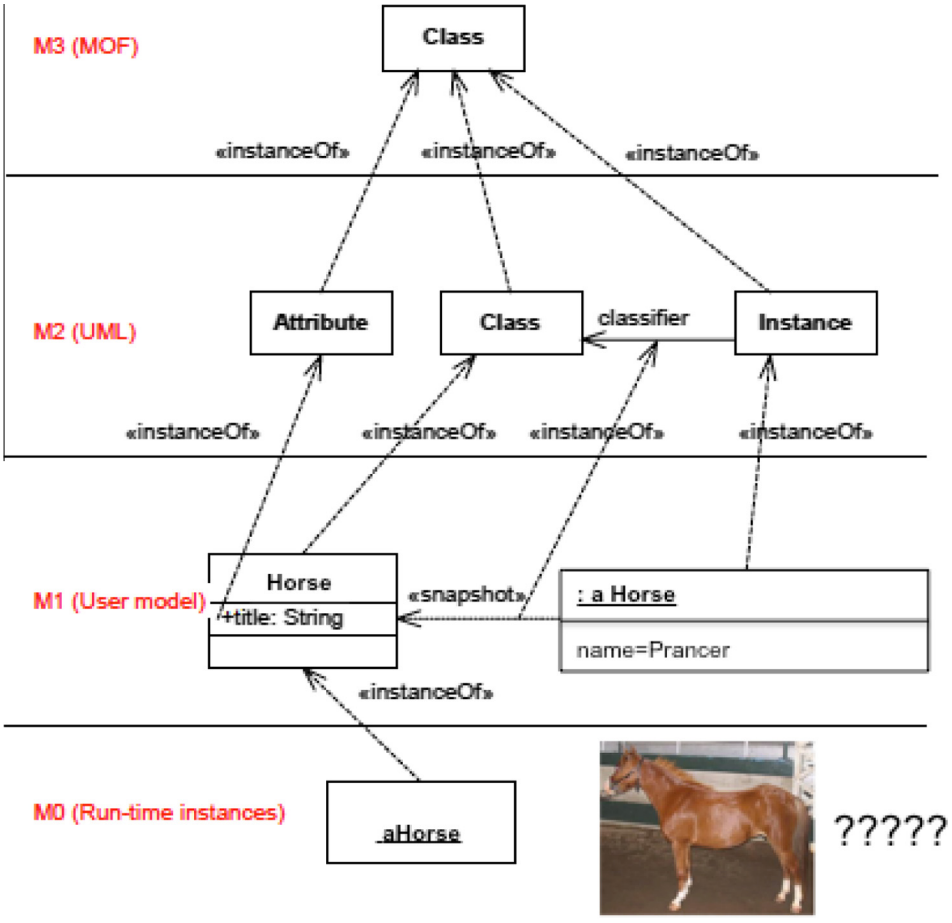

Fig. 1. Four-layer hierarchy of the Object Management Group (OMG).