The Journal of Systems and Software 83 (2010) 2591–2606

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Domain-specific language modelling with UML profiles by decoupling abstract and concrete syntaxes

Jesús Pardillo*, Cristina Cachero

Department of Software and Computing Systems, University of Alicante, P.O. Box 99, E-03080, Spain

ARTICLE INFO

Article history:
Received 15 December 2009
Received in revised form 11 August 2010
Accepted 11 August 2010
Available online 19 August 2010

UML
Diagramming
Modelling
Profiles
Syntax
Visual languages

ABSTRACT

UML profiling presents some acknowledged deficiencies, among which the lack of expressiveness of the profiled notations, together with the high coupling between abstract and concrete syntaxes outstand. These deficiencies may cause distress among UML-profile modellers, who are often forced to extend from unsuitable metaclasses for mere notational reasons, or even to model domain-specific languages from scratch just to avoid the UML-profiling limitations.

In order to palliate this situation, this article presents an extension of the UML profile metamodel to support arbitrarily-complex notational extensions by decoupling the UML abstract and concrete syntax. Instead of defining *yet another metamodel* for UML-notational profiling, notational extensions are modelled with DI, *i.e.*, the UML notation metamodel for diagram interchange, keeping in this way the extension within the standard. Profiled UML notations are rendered with DI by defining the graphical properties involved, the domain-specific constraints applied to DI, and the rendering routines associated. Decoupling abstract and concrete syntax in UML profiles increases the notation expressiveness while decreasing the abstract-syntax complexity.

1. Introduction

UML profiling (Object Management Group, 2009) is a straightforward technique to extend UML. UML profiles allow modellers to adapt UML to fit the representational needs of particular domains. For this purpose, they define *stereotypes* (Wirfs-Brock et al., 1994) that, applied over the UML *metaclasses*, redefine their notation, syntax and semantics (Berner et al., 1999). Therefore, the core extension concept is the (meta-) metaclass `Stereotype`.[1] This metaclass specialises the metaclass `Class`, to which it adds `Properties` (called *tagged definitions*) and `Constraints` (by means of the `Namespace::ownedRule` property), which are usually defined in OCL (Object Management Group, 2006a). Stereotypes are related with the corresponding metaclasses by means of `Extensions` (Object Management Group, 2009).

UML profiles also support notational extensions through arbitrary `Icons` associated to `Stereotypes`. These `Icons` decorate the notation of extended metaclasses, increasing diagram comprehensibility (Staron et al., 2006). The ulterior representation of these icons is managed by modelling tools according to a set of predefined rules, defined by UML (Object Management Group, 2009) (p. 674): *e.g.*, `Classes`, which are diagrammatically represented by a box containing various compartments, can be represented by a mere icon *iff* a single stereotype is applied to the class and class properties are hidden.

Moreover, stereotypes have a certain meaning associated. Unfortunately, UML does not properly formalise (in the mathematical sense) many of its metaclass semantics (Kong et al., 2009; Opdahl and Henderson-Sellers, 2002). Therefore, the stereotypes semantics is usually expressed (like the UML semantics itself) as an informal description in natural language.

Depending both on their base metaclass and their complexity, stereotypes can be classified into: *decorative* (concrete-syntax[2] extension), *descriptive* (abstract-syntax extension), *restrictive* (descriptive with syntactic constraints) and *redefining* (removal of the original UML syntax) (Berner et al., 1999). Some authors refer to the UML profiles that only contain decorative, descriptive or restrictive stereotypes as *conservative extensions* of UML, while UML profiles that contain redefining stereotypes are called *non-conservative* extensions (Turski and Maibaum, 1987). This distinction is important because only conservative extensions are 'safe', in the sense that they avoid undesired side-effects in UML

* Corresponding author. Tel.: +34 965 90 3400x2075; fax: +34 965 90 9326.
E-mail addresses: jesuspv@ua.es, jesuspv@dlsi.ua.es (J. Pardillo), ccachero@dlsi.ua.es (C. Cachero).

[1] The reason why `Stereotypes` can be considered meta-metaclasses is that they are defined over UML metaclasses.

[2] Concrete syntax and notation are herein used as synonyms. The notions of both concrete and abstract syntax are defined in Appendix A.

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.08.019

models. The reason is that conservative extensions abide by a variant of the Liskov's *substitution principle* (Liskov, 1987), which makes possible to substitute an instance of a stereotyped metaclass by an instance of the extended metaclass without incurring in syntactic inconsistencies (although, of course, with this substitution we may incur in a expressiveness loss). This means that modellers should try to come up with conservative extensions whenever possible. Unfortunately, in the last years we have witnessed how many of the myriad of UML profiles proposed in literature are still non-conservative. This situation is often due to the fact that mere notational reasons lie behind the selection of many extended metaclasses in existing UML profiles (Section 2). Choosing base metaclasses for notational reasons also often provokes the unnecessarily complex redefinition of UML elements in order to support the domain-specific syntax.

This article dives into this situation and argues that both this complexity and the problems caused by non-conservative profiles could be alleviated if UML decreased the coupling between profiled concrete and abstract syntaxes. This decoupling would allow designers to extend from metaclasses whose abstract syntax is closer to the concept represented by the stereotype, without being forced to also use its notation. In other words, modellers should not be forced to extend from a given metaclass in order to be able to use its notation.

Going one step further, whenever it is possible to find (a) closer matches between representation and representee notations or (b) aesthetically more pleasant representations than the ones provided by the general-purpose UML notation, modellers should not be forced to stick by the UML notation, since appropriate notations foster a more effective diagrammatic communication (Green and Petre, 1996).

Many modelling tools, being aware of this fact, have already incorporated their own alternatives to the UML 'icon-based extension rules' (Object Management Group, 2009) (p. 674). For example, tools like the award-winning MagicDraw[3] allow to enrich the notation of UML `Associations` with additional properties, such as end shapes, colour, and thickness, to cite a few. The decoupling between abstract and concrete syntax in UML profiles would make these 'ad-hoc' solutions unnecessary. Otherwise stated, UML would benefit from allowing for a standard way to enrich notations independently from abstract syntax or semantics.

This article is organised as follows. Next section (Section 2) dives into the problems that derive from the high coupling between abstract and concrete syntaxes in UML profiles. In particular, Section 3 reviews the case of UML profiling for the data-warehousing domain. These problems are further discussed by means of an ER modelling running example (Section 4). Section 5 outlines the foundations behind the UML notation, namely the DI metamodel. Section 6 presents how it is possible to solve some common profiling problems by using this DI metamodel, and illustrates the approach by applying the proposed solution to the aforementioned ER running example. Section 7 completes the approach with a prototype implementation that shows how a DI-based UML profiling tool may work in practice. Last, Section 8 discusses the presented solution and its implications for UML profiling.

2. Abstract and concrete syntax coupling in UML profiles

During the definition of UML profiles, modellers need to make decisions on the ideal candidates for extension. Such suitability is usually decided based either on the abstract syntax or on the notation of the profiled concept. Whatever the criterion, the modeller usually chooses the base metaclass whose either abstract syntax or notation most closely matches that of the profiled concept.

During this process, the modeller can be faced with two extension situations that are prone to causing problems later on:

- The modeller chooses a base metaclass whose abstract syntax is similar to that of the profiled concept. This base metaclass, however, provides a notation that greatly differs from the desired one, and whose adaptation requires more than the mere addition of icons. Since this icon addition is the only normative notational extension in UML, the resulting notation is not suitable for the domain, and therefore the profile is dismissed in practice.
- The modeller chooses a base metaclass whose notation (concrete syntax) is similar to that of the profiled concept. This base metaclass, however, needs to be heavily constrained in order to represent the abstract syntax of the profiled concept, what unduly increases the complexity of the profile and turns it into a non-conservative extension.

Let us exemplify these situations.

The first situation occurs when a metaclass is extended for syntactic reasons, but its notation (which has to be adopted – decorated or not with icons – too) is unsuitable for the domain-specific language.

Example 2.1 *(ER profile for conceptual data modelling).* Below in this paper, we present as a running example different versions of an ER profile (Section 4). Fig. 4 presents the version where the most suitable candidates from a syntactic point of view have been selected. The resulting notation can be seen in the right side of the figure. This iconised class-diagram notation is clearly different from the look one would expect to find for an ER diagram.

The second situation occurs when a metaclass is extended because of its notation suitability, at the price of having to deal with a syntax that does not correspond with that of the target concept. When this happens, it is common to define an extra set of constraints over the profiled UML metamodel in order to adapt such syntax. Constraints that redefine the syntax of base UML metaclasses, violating the syntax rules of the base metaclass, not only increase the profile complexity, but they also often turn it into a non-conservative extension, much more error-prone for designers.

Example 2.2 *(UML profile for data warehouses by Abelló et al. (2006)).* Abelló et al. (2006) present a domain-specific language for data warehouses. In this UML profile, facts and dimensions of analysis are modelled as `Classifier Stereotypes`. Such mapping allows the modelling of domain-specific relations (*e.g.*, aggregations, data flows, etc.) between them. UML `Classifiers` are a 'classification of instances' (Object Management Group, 2009) (Section 7.3.8). However, in this UML profile, neither facts nor dimensions should have instances; in fact, syntactically speaking, both notions are closer to containers than to classifiers. This profile is a good example of a non-conservative extension. This UML profile still poses another notational problem: the `Dimension Stereotype` has two notations. At a higher level, dimensions are denoted as rectangles with a text label (the `Classifier` name) inside. At a lower level, they are denoted as rectangles containing several classes which they are composed of. This contrasts with UML, where the `Classifier` is an abstract model `Element` and so, properly speaking, it should have no notation (Object Management Group, 2009) (Section 7.3.8).

This situation can become even worse. Sometimes the UML notation, even with the addition of icons, is simply not expressive enough to denote elements of the domain-specific language. As an example we can cite how, in case of scalability concerns, specific graph layouts (Eichelberger and Schmid, 2009; Dobing and Parsons,

[3] http://www.magicdraw.com