Information Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Information Systems

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

A multi-level approach to modeling language extension in the Enterprise Systems Domain

Colin Atkinson [a], Ralph Gerbig [a,*], Mathias Fritzsche [b]

[a] University of Mannheim, Germany
[b] SAP AG, Germany

ARTICLE INFO

Article history:
Received 28 February 2014
Received in revised form
22 September 2014
Accepted 14 January 2015

Keywords:
Multi-level modeling
Model language extension
Orthogonal classification architecture
Linguistic classification
Ontological classification

ABSTRACT

As the number and diversity of technologies involved in building enterprise systems continues to grow so does the importance of modeling tools that are able to present customized views of enterprise systems to different stakeholders according to their needs and skills. Moreover, since the range of required view types is continuously evolving, it must be possible to extend and enhance the languages and services offered by such tools on an ongoing basis. However, this can be difficult with today's modeling tools because the meta-models that define the languages, views and services they support are usually hardwired and thus not amenable to extensions. In practice, therefore, various work-arounds have to be used to extend a tool's underlying meta-model. Some of these are built into the implemented modeling standards (e.g. UML 2, BPMN 2.0 and ArchiMate 2.0) while others have to be applied by complementary, external tools (e.g. annotation models). These techniques not only increase accidental complexity, they also reduce the ability of the modeling tool to ensure adherence to enterprise rules and constraints. In this paper we discuss the strengths and weaknesses of the various approaches for language extension and propose a modeling framework best able to support the main extension scenarios currently found in practice today.

1. Introduction

Over the last few years the success of open modeling frameworks such as the Eclipse Modeling Framework (EMF) [1] has lead to a significant increase in the number of tools driven by meta-models rather than "hardwired" data types. This in turn has established a thriving industry around such tools offering extensions to their core modeling languages. This capability is particularly important for visualizing enterprise systems with modeling frameworks such as ArchiMate [2], The Open Group Architecture Framework (TOGAF) [3] and the Zachman framework [4], since the number of different views and stakeholders is large and heterogeneous.

Sometimes the expressive capabilities of a modeling language used to build a tool can be extended using extension mechanisms already built into the language such as those of the UML 2 [5], BPMN 2.0 [6] and ArchiMate 2.0 [2] extension mechanisms. Often, however, the modeling languages upon which tools are based do not offer built-in extension mechanisms, or if they do, these mechanisms are not fully implemented by standard-compliant tools. In such cases, other techniques such as model annotation via model weaving have to be applied to store the additional information needed by the extended language. The additional information is stored in separated annotation models which are linked to the main model by using model weaving techniques of the form described by

* Corresponding author.
E-mail addresses: atkinson@informatik.uni-mannheim.de (C. Atkinson), gerbig@informatik.uni-mannheim.de (R. Gerbig), mathias.fritzsche@sap.com (M. Fritzsche).

http://dx.doi.org/10.1016/j.is.2015.01.003
0306-4379/

Bézivin et al. [7]. Model weaving essentially defines the technique used to store links between two models.

In practice, therefore, software engineers and enterprise architects are often faced with a range of different options for extending the capabilities of the languages supported by modeling tools, each with a different mix of advantages and disadvantages. Although one might imagine it is always best to use the built-in features to extend a modeling language, this is not always the case. Ad hoc model extension techniques can often lead to extension definitions which are better structured and of higher quality in terms of such software engineering maxims as "separation of concerns", "high cohesion" and "loose coupling". In fact, at the time of writing, we believe no existing modeling framework offers the ideal mix of features needed to support the full range of modeling language extension requirements found in the enterprise computing domain. The goal of this paper is to address this problem by describing what this mix of features should be and what properties a modeling framework should have to support them.

In the next section we first identify the different fundamental strategies for supporting modeling language extension and characterize their strengths and weaknesses. In Section 3 we then present these strategies in the context of a small example scenario from the domain of business process performance modeling. This example highlights the need for new modeling approaches supporting modeling language extension. Section 4 then introduces such an alternative modeling approach, known as multi-level modeling, which we believe provides the optimal approach for model extension, while Section 5 analyses this architecture from the point of view of the strategies and requirements outlined in previous chapters. It also proposes enhancements to current multi-level modeling approaches to address some identified weaknesses. Section 6 then illustrates how this extension-aware form of multi-level modeling could be applied in an industrial setting. Then Section 7 evaluates the proposed multi-level modeling approach on a real world scenario to show feasibility of the approach followed by related work in Section 8. Finally, Section 9 concludes with some final remarks and suggestions for future work.

This paper is an extended and reworked version of a previous paper presented at the EDOC 2013 conference entitled "Modeling Language Extension in the Enterprise Systems Domain" [8]. In contrast to the original paper, this version has a dedicated Evaluation section (Section 7), presenting the strengths of the approach in the context of a significant real-world example from the literature, and a Related Work (Section 8) section. The section about model annotation (Section 2.3) has been extended using an aspect-oriented modeling implementation method and the description of multi-level modeling (Section 4) has been significantly extended to better introduce the approach.

2. Modeling language extensions

When extending a modeling language it is useful to distinguish between language enhancement and language augmentation. The former focuses on extending a language with additional modeling concepts from the same domain as the original concepts, while the latter introduces new concepts from a different problem domain than those in the original language. An example of language enhancement is adding an additional kind of class to the UML meta-model so that users can instantiate special classes (e.g. Java beans) in addition to regular, unspecialized classes. This effectively enriches the original language with concepts that make it more expressive in its original domain. An example of language augmentation, on the other hand, is to extend a business process modeling language with data for performance simulation. This effectively enriches the original language with the ability to express concepts from a completely different domain.

The modeling tools and frameworks available today essentially support three fundamental approaches for modeling language extension — (a) dedicated, built-in extension mechanisms (b) meta-model customization and (c) model annotation. Each has pros and cons when used for practical enhancement and augmentation tasks. Choosing the wrong mechanism for an extension task can easily break design principles, such as separation of concerns, loose coupling and high cohesion, and introduce accidental complexity [9,10] into models. These different approaches are elaborated further and their pros and cons are highlighted in the following subsection.

2.1. Meta-model customization

The language extension approach that at first sight appears to be the most straightforward is to directly change the language's meta-model. However, in practice this turns out not to be the case with most frameworks and tools available today because either the meta-models are hardwired and not accessible for changes at run-time, or the language extensions defined by changing the meta-model cannot directly be used in the modeling tool. In the second case, after a meta-model has been customized the modified tool needs to be recompiled and redeployed to make it aware of the changes. This is not only a tedious and error prone task it often has to be followed up by the use of model migration tools to keep the model instances in-sync with the changed meta-model.

Another problem of meta-model customization is that uncontrolled tinkering with a meta-model, at any place in its inheritance hierarchy, can easily lead to violations of the separation of concerns design principle and lead to meta-models with low cohesion. When mixing meta-model-elements from two different problem-domains (e.g. from the business process and simulation domains) it is important to integrate them in a way that respects high cohesion and loose-coupling, otherwise the resulting meta-model can quickly become unmaintainable. This is why, when evolving UML 1.0 into UML 2.0, the OMG put so much effort into separating concerns by organizing model-elements into packages. A concept similar to packages is thus an essential prerequisite for augmenting a language through meta-model customization in a maintainable way. A weakness of most current package mechanisms used in meta-modeling tools is that all defined packaging are always present. It is usually not possible to create customized versions of a tool, with

Please cite this article as: C. Atkinson, et al., A multi-level approach to modeling language extension in the Enterprise Systems Domain, Information Systems (2015), http://dx.doi.org/10.1016/j.is.2015.01.003