Information Systems 54 (2015) 289–307

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

A R T I C L E I N F O

Article history:
Received 28 February 2014
Received in revised form
22 September 2014
Accepted 14 January 2015
Available online 23 January 2015

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

A B S T R A C T

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/© 2015 Elsevier Ltd. All rights reserved.

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 customized versions of the language that incorporates only a selected subset of the available packages. This can cause many problems including significant version and configuration management issues when modelers use language features that are not supposed to be used for specific tasks and specific views.

2.2. Language built-in mechanisms

As previously mentioned, most existing modeling tools, especially in the enterprise computing domain, do not support the direct customization of their underlying meta-model, but instead offer special "built-in" mechanisms that allow the effect of meta-model customization to be achieved without actually changing it (since it is hard-wired). Examples of such techniques are the UML profiling mechanism (based on stereotypes), and the BPMN 2.0 and ArchiMate extension mechanisms.

Two problems are immediately obvious with such approaches. The first is when each tool in a heterogeneous enterprise computing environment defines its own non-standard extension mechanisms, significant compatibility and interoperability problems arise. For instance, it can easily occur that a "simulation" extension created using the BPMN extension mechanism cannot be used with an extension mechanism defined in another business process modeling language (e.g. ArchiMate business processes) even though they are conceptually compatible. The second issue is that some tools do not fully implement modeling standards, and it is not uncommon for tools to provide no support for the built-in extension mechanisms of the language they are based on. An example for such a case is the UML which defines four different levels of compliance. UML Profiles only need to be supported by tools starting at the third of four compliance levels. Another example is the ArchiMate 2.0 extension mechanism which is not necessarily supported by all implementing tools. The features which are covered by the standard implementing tools are centrally listed in the so called "AchiMate Tool Certification Register" available at [11].

To use a language's built-in extension mechanism a user has to create his own set of (logical) meta-model-elements in a separate location and connect them with the core model-elements in the hardwired meta-model. This fits the language augmentation scenario because it is desirable to separate the concepts from the two different domains – the core modeling language concepts and the augmentation concepts – in order to separate concerns. Moreover, users of a tool that supports language augmentations are not usually forced to include all the augmentations that have been defined, which is the case with meta-model customizations. The different augmentation packages can usually be "plugged in" or "plugged out" of the modeling environment at any time without the need for any recompilation or deployment.

Built-in extensions can obviously support language enhancement scenarios but not without extra accidental complexity. In many cases it is necessary to add new meta-model-elements that replicate information already available in existing (hardwired) meta-model-elements because of the lack of direct meta-model editing capabilities. To modify an attribute, for example by renaming it, a new meta-model-element needs to be created which then adds the modified attribute. This not only leads to duplication of information – once in the original language and once in the extension, it also makes the language more difficult and complex to use. Contemporary built-in extension mechanisms therefore have weaknesses for language enhancement scenarios but provide reasonable support for model augmentation (provided that they are supported by tools, which is often not the case).

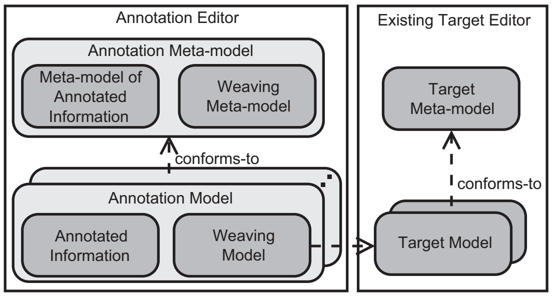


Fig. 1. A modeling language extended through annotations.

2.3. Model annotation

In the previous two cases, the mechanisms used to define extensions are defined as part of the language to be extended, even if they are not always supported by tools. With model annotation approaches, however, both the language used to define the extension, and the language used to attach instances of the extension to instances of the core domain concepts, can be completely different. For example, [12] presents a light weight model annotation approach that allows structured information (see Annotated Information in Fig. 1) to be attached to other model-elements in a Target Model. This linkage is technically achieved by refining the Weaving Meta-Model defined in [13] with an Annotation Meta-Model. Weaving models permit links between model-elements to be defined via Unified Resource Identifiers (URI) which are stored in the weaving model. A big advantage of this approach is that the extending annotation model does not pollute the target model and therefore does not violate the design principle of separation of concerns. Moreover, like extensions created using built-in extension mechanisms, annotations can be included or excluded as required at any time during the modeling process. Most significantly, since the approach is independent of a language standard or tool, it can be applied in any kind of modeling framework or enterprise computing environment. Thus, an augmentation originally defined for BPMN can, for example, be reused with the Business Process Execution Language (BPEL) [14], UML Activity diagrams, ArchiMate diagrams or Event-driven Process Chains (EPC) [15].

Given the benefits of this approach, Annotation Editors have even been defined to simplify the definition of annotations in [16]. Such editors provide basic annotation capabilities without modifying Existing Target Editors. This, however, also implies that two separate tools need to be employed by the user, one for defining the target model and one for expressing the annotated information. If the goal is seamless integration of the two editors, for example

		Extension Scenario		
		(1) Enhancement	(2) Augmentation	(1) and (2)
Extension Mechanism	Built-in	- introduction of accidental complexity through meta-model element duplication	+ separation of concerns by separating problem domains of host language and augmentation	+ Pluggable extensions - not standardized across tools and modeling languages
	Meta-model Customization	+ Direct change of enhanced meta-model elements	- mix of augmentation domain and host language domain, because of hard wired packages	+ At first glance, very simple - not supported by most state-of the art meta-modeling frameworks - Can break existing toolings - Can cause redeployment
	Model Annotation	- introduction of accidental complexity through meta-model element duplication	+ separation of concerns by separating problem domains of host language and augmentation	+ indepenent of tools or modeling languages + Pluggable extensions - Tools need to be customized for good integration

Fig. 2. Suitability of different forms of Extension Mechanisms. Plus means advantage, whereas minus means disadvantage.

when an annotation should impact the layouting or behavior of an existing target editor, the target editor has to be modified.

The implementation of model annotations by using Aspect-oriented Modeling (AOM) got more and more attention in the past. AOM originates from the idea of Aspect-oriented Programming (AOP) [17] and the languages supporting it (e.g. AspectJ [18]). In [19] two approaches supporting AOM are identified. The first is to combine two models by common model-elements e.g. an id. The second is to use concepts from AOP languages at the model level, e.g. define join points in models and use before, after and around advices. Both approaches have in common that two kinds of models are created and combined — so called base models and aspect models. The base models define the core of a modeling language and are connected to aspect models which define different extensions to this base model. Connections between base and aspect models have to point from the aspect to the base model. AOM is widely applied. One big field of application is reuse. For this purpose a reusable-aspect model (RAM) [20] has been developed which has been extended in various domains, e.g. security [21] or concern-oriented software design [22]. The application area relevant for this work is modeling language extension as described in [23]. In [23] AOM is used to define extensions, covering e.g. reliability properties, to a base language which is the Palladio Component Model [24]. Only basic concepts of AOM are used but more powerful constructs like extensions which are only applied if certain conditions hold true etc. can be envisaged. This would lead to a transition from the first of the two earlier described AOM classifications to the latter using advanced features such as the ones available in AOP languages like AspectJ.

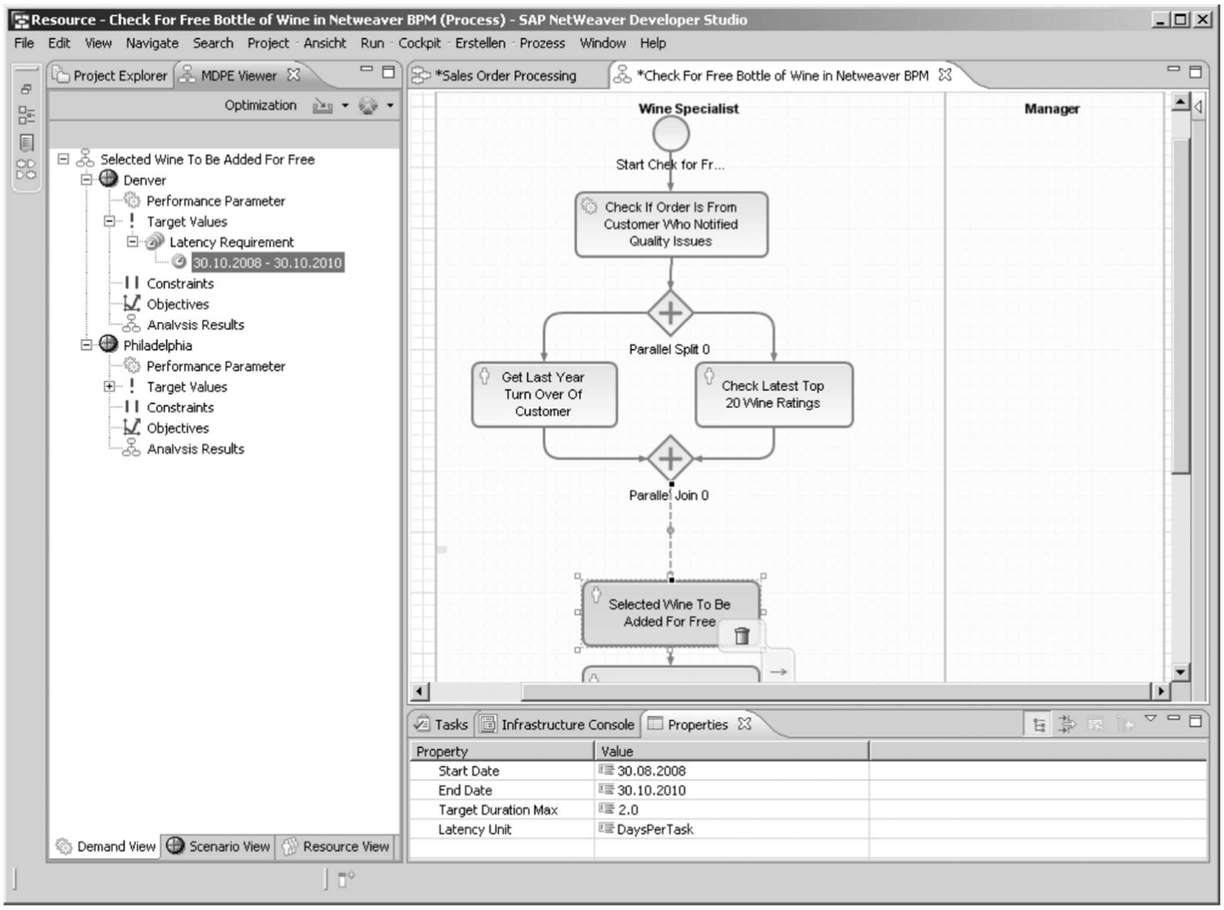
The main usage scenario for the model annotation approach is modeling language augmentation because a modeling language can be easily augmented independently of modeling languages and tools. Model annotation can also be used for modeling language enhancement, but the same issues arises as with the use of built-in language mechanisms. Unnecessary, duplicated model-elements often have to be introduced which are not needed when employing the meta-model customization approach.

2.4. Summary

Fig. 2 briefly summarizes the pros and cons of the here discussed extension mechanisms, as supported in current modeling frameworks, from the point of view of language enhancement and language augmentation scenarios. The rows define the three extension approaches identified above (language built-in mechanisms, meta-model customization and model annotation). The columns define how these approaches perform in the language extension scenarios (enhancement and augmentation). The last column, titled (1) + (2), lists the properties which are valid in both extension scenarios, enhancement and augmentation. Positive properties are identified by a leading + and negative ones by a leading −. In summary it can be observed that no extension mechanism fits all extension scenarios well. Hence, the choice of an appropriate extension mechanism must be carefully evaluated. A wrong choice can lead to increased accidental complexity (e.g. duplication of model-elements), violation of the separation of concerns design principle, extensions which are not compatible between modeling environments, loss of usability (e.g. two different model editors need to be used for extension and host language) and existing modeling editors to not work anymore with the extended language (e.g. extended meta-model is not supported anymore). The real-world example in the following sections shows how such an evaluation of extension mechanism can look like.

3. Augmentation of business process modeling tools with performance information

To illustrate a typical language extension scenario we will consider the problem of implementing a business process performance engineering environment called Model-Driven Performance Engineering (MDPE) [25] which supports process performance analysis based on process simulations, optimizations, etc. This goal is achieved by extending a third party, closed source tool for business process modeling. To apply process performance analysis approaches to a business process, more information than usually stored by business process modeling languages is needed. Moreover, this information comes from a different domain than the domain of


Resource - Check For Free Bottle of Wine in Netweaver BPM (Process) - SAP NetWeaver Developer Studio
File Edit View Navigate Search Project Ansicht Run Cockpit Erstellen Prozess Window Help
Project Explorer
MDPE Viewer
Optimization
Selected Wine To Be Added For Free
Denver
Performance Parameter
Target Values
Latency Requirement
30.10.2008 - 30.10.2010
Constraints
Objectives
Analysis Results
Philadelphia
Performance Parameter
Target Values
Constraints
Objectives
Analysis Results
Demand View
Scenario View
Resource View
*Sales Order Processing
*Check For Free Bottle of Wine in Netweaver BPM
Wine Specialist
Manager
Start Chek for Fr...
Check If Order Is From Customer Who Notified Quality Issues
Parallel Split 0
Get Last Year Turn Over Of Customer
Check Latest Top 20 Wine Ratings
Parallel Join 0
Selected Wine To Be Added For Free
Tasks
Infrastructure Console
Properties

Property	Value
Start Date	30.08.2008
End Date	30.10.2010
Target Duration Max	2.0
Latency Unit	DaysPerTask

Fig. 3. Screenshot of the business process performance case study implementation.

business process modeling and execution, the domain of business process simulation. Thus, the core language used to define the business processes must be augmented to handle the performance related information.

In principle, any one of the three aforementioned extension approaches could be used - meta-model customization, use of built-in extension mechanisms and model annotation. However, in this case, because the chosen modeling environment is a closed-source, third party tool with a hardwired meta-model the first approach (meta-model customization) is not an option. The second option (the use of built-in extension mechanisms) is not an option either because the tool is an Eclipse plugin (implemented using the EMF and its meta-modeling language Ecore, an implementation of the EMOF standard) which has no built-in extension mechanism. Moreover, although the tool supports the BPMN 2.0 language, it does not support this language's built-in extension mechanism since it is not required to do so to be compliant with the standard which allows different levels of compliance in [6]. As is often the case, therefore, the only available option in this scenario is to apply the third approach (model annotation).

Fig. 3 shows a screenshot of the implementation using annotation models developed by Fritzsche [16]. In this screenshot, the BPMN based editor NetWeaver BPM [26] (see upper right view) has been extended with an annotation editor. The implementation adds a view to Eclipse which displays the content of the business performance annotation model (see left). The modeled process in the screenshot defines a workflow for adding free bottles of wine to orders of those customers which had a high turnover in the last year. The bottle of wine is added by a so called Wine Specialist as shown by the left process lane. In order to investigate how many wine specialists are required to execute the process, a process simulation can be employed based on additionally annotated data.

The annotated data added by the annotation editor includes the so called Latency Requirement of two days per task for the process step Select Wine To Be Added For Free. This means that the step needs to be executed within maximally two days. Such annotations are useful to evaluate process simulation results such as determining the minimum number of wine specialists required to meet the two day threshold. This model is connected to business process modeling tools through various connectors. For each modeling editor that is connected with the provided MDPE view a new connector needs to be developed. That is all that needs to be done to connect the language augmentation to a new modeling language. This has the advantage that the MDPE view can show different information depending on what is selected in the model editor. However it has the disadvantage that the used host modeling environment must have an extensible user interface that can be extended by a view of the additional data. If such an extensible interface is not available a second application, an annotation model editor, is needed in order to edit the annotation model.

In a similar case study Rodriguez et al. [27,28] show how the other two extension mechanisms can be used to augment business process models represented using BPMN diagrams and UML Activity diagrams with security related information. In the first case they achieved this by customizing the BPMN meta-model [27], and in the second case they used the UML's built-in extension mechanism based on stereotypes [28]. This, however, results in two mutually incompatible extensions for Activity Diagrams and BPMN diagrams.

From the two case studies it becomes obvious that current available extension mechanisms have different problems when applied in practice. Model annotation approaches require an extensible modeling tool or a separate annotation model editor, while the work of Rodriguez et al. [27,28] lead to mutually incompatible extensions of UML and BPMN. In the following sections a technology overcoming these weaknesses will be introduced and demonstrated on the example of business performance engineering.

4. Multi-level modeling

As observed in the previous sections, the different practical approaches for extending modeling languages in the domain of process modeling and enterprise computing have different strengths and weaknesses. Some of these (e.g. what can in principle be varied and what not) are inherent to the approach, while others (e.g. what is implemented and what not) are reflections of the practical choices made by particular tool vendors. In general, however, it would clearly be beneficial to have a modeling framework, and related modeling tools, that are able to avoid all the weaknesses and combine all the strengths of these approaches. To achieve this a modeling infrastructure is needed which combines the stability and universality of an agreed standard (that can be hardwired into tools) with the flexibility of a “soft” meta-model that can be extended without having to recompile and redeploy supporting tools before the extended languages can be used.

Such an infrastructure is provided by the so called multi-level modeling approach which is based on the Orthogonal Classification Architecture (OCA) [29]. The key innovation in this architecture is to organize model-elements according to two completely orthogonal dimensions of classification—one dealing with linguistic classification and the other dealing with domain (i.e. ontological) classification. An example of the orthogonal classification architecture is shown in Fig. 4. The linguistic levels $L_2 - L_0$ are comparable to the four layer architecture ($M_3 - M_0$) as shown in the UML [30]. L_2 contains the multi-level modeling language meta-model that spans all ontological levels. Meta-model-elements are Inheritance (not displayed in Fig. 4), Clabject, Method, Level, Feature etc. These meta-model-elements have linguistic attributes called traits to better distinguish ontological attributes from linguistic attributes. Examples of traits are potency in the case of Clabject or durability in the case of Attribute. In this example the linguistic classification levels are stacked vertically while the ontological levels are stacked horizontally within the L_1 linguistic levels. Real world objects are placed at L_0 for the concepts of a BPMNTask, a conceptual representation of a BPMNTask is shown with ??? as title to show that a general concept that needs to be instantiated and filled with information is defined. This concept is instantiated with a real world Fill In Post Label task as it would occur on a business process diagram. The task is then instantiated by a person sitting at the desk executing the task by filling in a post label.

Although the figure shows only three ontological modeling levels, the modeler can choose an arbitrary number. Moreover, all ontological levels are soft and changeable at the same time, since they are all just data from the point of view of the linguistic (meta-)model L_2. Changes to one ontological level will immediately effect all other levels—a

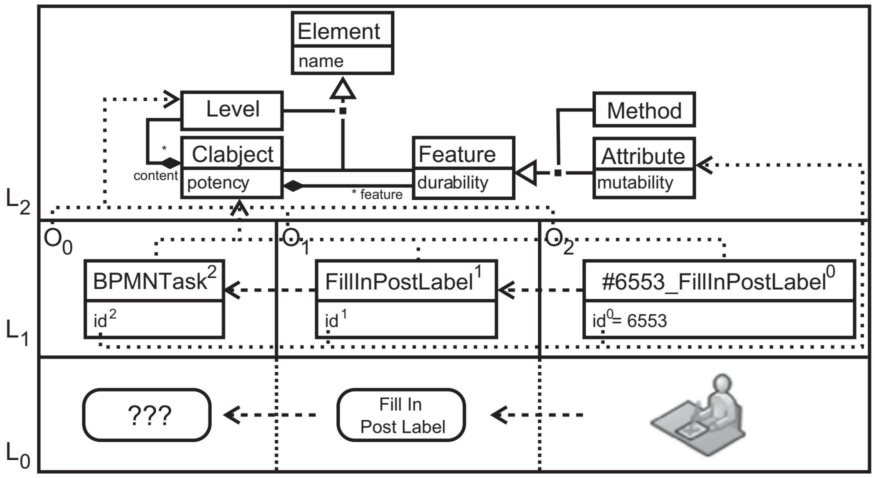


Fig. 4. Orthogonal classification architecture instantiated with a business process model.

characteristic that has been called live meta-modeling [31]. Besides the standard meta-modeling capabilities, multi-level modeling allows domain-specific modeling languages (DSL) to be defined within ontological modeling levels rather than through extension mechanisms in the linguistic meta-model. It is possible to define domain-specific graphical or textual representations on one level and completely or partially [32] display the level classified by it using this representation.

Fig. 4 also shows that all model-elements have a numeric superscript next to their name. This value, known as the element's potency, is used to support a concept known as deep classification. The potency of a model-element states how many levels it can influence (i.e. over how many levels it can spawn instances). The potency of a model-element can be either a non-negative integer value or $*$. In the former case the instances must have a potency that is one lower than the type's potency. Model-elements with potency 0 cannot have instances. In the latter case, which is not restricting the depth of the classification tree, instances can have either $*$ potency or a non-negative integer value. Attributes also have a potency as superscript. This potency influences over how many instantiation steps an attribute can endure (i.e. handed over to instances). Thus this potency is called durability. Attributes have an additional potency, called mutability, which indicates if a value can be changed at the clabject's instances. In the example, the mutability is not shown by convention as it is the same as the attributes durability. If an attribute has mutability 0 its value cannot be changed anymore at the clabject's instances. It can, however, still be passed on to instances if the durability is higher than 0. Fig. 4 shows all model-elements at level O_0 with potency 2 and durability 2 and their instances at level O_1 with potency 1 and durability 1. The model-elements at O_2 have a potency of 0 and a durability of 0 which is one lower than the potency and durability of the types at O_1.

The figure also shows that all classifying model-elements are instances of a single linguistic model-element, Clabject. The term “clabject” originates from a combination of the terms “class” and “object” which reflects the class/object duality of model-elements in the middle levels. These model-elements are an instance (object) of their types and at the same time type (classes) for their instances at the following levels. The attributes are instances of the linguistic Attribute model-element at level L_2.

One of the advantages of the OCA is that all kinds of information beyond just domain information can be described in a multi-level way. For example, clabjects can each have a visualizer which specifies how the clabject, as well as its subtypes and its instances, are rendered. This is achieved by applying a special visualization search algorithm when a clabject is rendered which first looks for an appropriate visualizer attached to the clabject itself, and if none is available, searches up its inheritance hierarchy. If no visualizer is found at the clabject's level the next level is searched starting from the clabject's ontological type(s). This is repeated on all ontological levels until a domain-specific visualizer is found. Finally, if no domain-specific visualizer is found on any ontological level, the general purpose notation for the linguistic type, residing at L_2, is used. This supports a general purpose concrete syntax which resembles the UML and Entity-relationship Model (ER) [33] concrete syntax. This algorithm can be used to override the visualization of existing modeling languages by defining a new domain-specific visualization for subtypes of standard model-elements contained in the original language. New model-elements instantiated from these subtypes can then use the new visualization instead of the default one by applying the visualization search algorithm. We refer to such languages, where one language is embedded arbitrarily within another and can use the concrete syntax of that other language as well as its own, as symbiotic languages [32].

Fig. 5 shows an enterprise computing scenario where this symbiotic capability can be very useful. On O_0 an excerpt of a BPMN-like domain-specific language is shown. Tasks can be connected with Tasks themselves or Splits. The clouds attached to AND and XOR show the symbols that have been defined for them using visualizers. In the figure, Level O_1 is shown twice (which would not usually take place in practice, of course). On the left the domain-specific concrete syntax is used whereas on the right a mix of domain-specific and general-purpose concrete syntax is used. Multi-level modeling tools such as Melanee [34], built on the principles previously described, make it possible for a modeler to toggle between general and domain-specific visualizations of the model, or individual model-elements, at the click of a button. This can be very useful for a domain novice who for instance is not familiar with the difference between the BPMN Splits, as shown in Fig. 5, which have a very similar notation (i.e. a rhombus with a differing symbol in the middle). Such a novice can easily switch to the general-purpose visualization at anytime to determine what kind of split is involved based on the classification information in the model-element's designator. In the example, one can imagine the novice toggles the domain-specific rendering of the central split clabject on the left hand side to the general purpose rendering on the right hand side, thereby discovering that it is an XOR split. A domain expert, on the other hand, can use the domain-specific notation at all times.

5. Language extension in multi-level modeling

Having introduced multi-level modeling and the capabilities that environments based on the OCA provide for

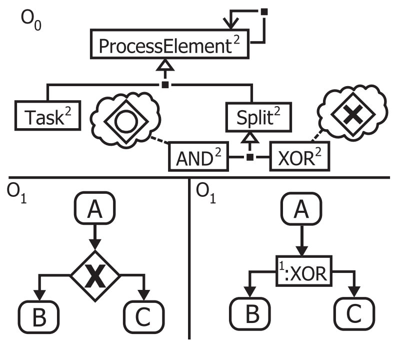


Fig. 5. Symbiotic language support.

defining domain-specific modeling languages, in this section we discuss how well it supports the two language extension scenarios described in the earlier part of the paper and what, if any, enhancements to multi-level modeling are desirable to optimize this support.

5.1. Modeling language enhancement

Tools based on the OCA provide powerful inherent support for the meta-model customization approach used for enhancement because all levels in an orthogonal classification hierarchy are "soft" and can be changed equally easily at any time, and every change immediately effects all other levels that are dependent upon it (live meta-modeling). This means that (a) the most abstract ontological levels (such as O_0 in Fig. 4) can be directly customized by a modeling language engineer and (b) that the effects of the customization are immediately "felt" and useable not only at the level below (O_1), but also at the level below that (O_2) and so on. Thus, no recompilation or redeployment of the modeling environment is needed to use the extended domain-specific language. To help users manage the impact of this power, Melanee provides an emendation service [31] which monitors the model for changes and updates all the levels to make them consistent after a change has occurred.

Fig. 6 shows schematically how the multi-level modeling approach inherently supports language enhancement through (meta-)model customization. The top of the level O_0 represents the core language (e.g. as described in a standard), while the bottom shows model-elements which enhance this language through inheritance (enhancement model-elements which are not part of the standard language are represented by a grey background). In this example, the general concept Task is specialized by ManualTask, PriorityTask and AutomaticTask using inheritance. This introduces the opportunity to model tasks which are either executed by a human (ManualTask) or a machine (AutomaticTask). A task executed by a human can be declared as being executed with priority (PriorityTask) within a defined time span to e.g satisfy reaction times agreed on in a service level agreement. Additionally the process modeling language's Task has been enhanced with a systemId attribute to allow it to be executed in a certain

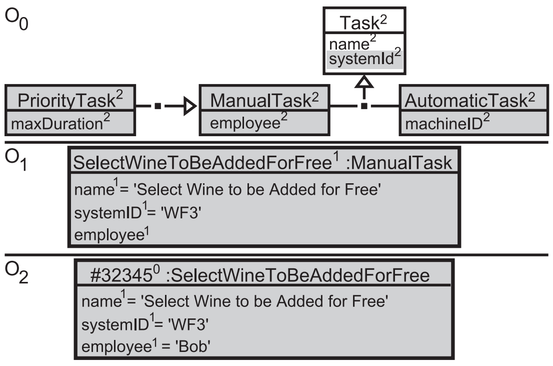


Fig. 6. A business process modeling language enhanced by a LatencyTask model-element and systemID attribute.

workflow engine requiring such an attribute. In Fig. 6 the attribute is directly added to the business process modeling language's meta-model. The figure also shows how the newly enhanced language can be used to immediately create an instance of a new abstraction (i.e. SelectWineToBeAddedForFree). Such statements can again be applied immediately to create further instances at the ontological level below (i.e. #32345). Note that with the vast majority of tools supporting meta-modeling customization (i.e. so called meta-modeling tools based on frameworks such as EMF), it would first be necessary to recompile and redeploy a new editor for the language in order to create instances of the new features such as ManualTask.

A problem that can arise during language enhancement is that there is potentially a gap of domain knowledge between the domain expert who initially created the language and the modeler adapting the language to the needs of the environment in which the enhanced language is to be used. Thus it can occur that a modeler breaks rules in a domain that cannot be captured by pure clabject modeling when enhancing the modeling language. For this case a deep OCL dialect is provided with Melanee [35] which is able to apply constraints to instances of clabjects as well as to the clabject on which constraint is defined and its subtypes. Hence, constraints can be defined on the type and classification levels meaning that one can define constraints which are also applied to subtypes. This for instance would help in a case where an "UrgentTask" is introduced as subtype of a PriorityTask with a maximum allowed duration of 5 hours. It can be assumed that it does not make sense to create subclasses of such an "UrgentTask" which allow an execution time with more than 5 hours. To prevent introducing such subtypes, constraints can be defined identifying such subclasses allowing more than 5 hours of execution time as invalid. The deep OCL dialect not only allows constraints to be defined on the ontological dimension but also on the linguistic one. This allows model-elements to be constrained in terms of potency, connections, naming, number of attributes and their durability etc. Hence, it would be possible to prevent the usage of star potency in a whole model or for certain subtypes of clabjects.

It can also happen that a modeler wants to enhance a modeling language in a way that certain constructs allowed in the host language are not allowed anymore on the enhanced language. Many motivations for such a scenario exist starting from legal regulations and company policies prohibiting certain things which can be modeled in the host language to limitations of tools which are configured with the created models. For such cases the OCL dialect can also be used to constrain an enhanced language.

The obvious question that arises is how and where built-in extension mechanisms fit into the multi-level modeling picture. There are two answers to this question depending on the perspective one wishes to take. One answer is that they do not fit in at all because they are completely redundant. Since language enhancement is fully supported by direct (meta-)model customization, with immediate availability, there is no need for any additional ad hoc extension mechanisms. The accidental complexity that would be introduced by adding them can thus be avoided. Provided that a moderately rich designation approach is available [36], everything that can be expressed or performed using stereotypes can also be expressed in multi-level modeling using direct (meta)-model customization. The other answer is that built-in extension mechanisms are “alive and well” in multi-level modeling, and in fact are already incorporated in an ubiquitous form, but not using ad hoc extension concepts such as stereotypes and tagged values, but in the form of the built-in specialization mechanism that is a core part of object-oriented modeling. In other words, the built-in extension mechanism of multi-level modeling environments is the ability to customize ontological model levels through specialization (i.e. inheritance). Although inheritance is only supported in a very basic form in Melanee at the moment, more advanced inheritance mechanisms (which can for example control which attributes and associations are inherited, as in [37]) can be implemented if they proof to be necessary in future. In short, (meta)-model customization is multi-level modeling's built-in extension mechanism. The key feature that makes this possible is that when interpreting specialization as a (meta)-model customization mechanism, one regards it from the perspective of the ontological levels, and when interpreting it as a built-in extension mechanism one regards it from the perspective of the linguistic levels. Thus, multi-level modeling allows modeling languages to be enhanced in a way that has less accidental complexity but equal power to traditional enhancement mechanisms such as UML stereotypes.

5.2. Modeling language augmentation

Multi-level modeling provides natural, inherent support for language enhancement, in its pure form, as currently supported by tools such as Melanee but it does not provide all the features necessary to support modeling language augmentation. As the previous example in Fig. 6 shows, minimalistic multi-level modeling environments are not able to support the separation of concerns principle that requires augmentation model-elements to be separated from core language elements. Nor can they support the requirement that augmentations can be included or excluded, as desired, on-the-fly. To support this a dynamic, multi-level-aware model composition mechanism needs to be added.

In this section we discuss what such a model composition mechanism, motivated by the annotation model and UML Profile approaches, could look like and how it can be seamlessly added to multi-level modeling. The purpose of an augmentation model is to define all the information needed to augment a modeling language focusing on one problem with language constructs focusing on a different domain. For modularity and flexibility reasons it should be possible to either directly embed such augmentation models into a multi-level model or load them over a network from a remote location. Additionally, one would like to toggle the view of a multi-level model to either apply the changes defined by an augmentation model or hide them. This provides modelers with the option of using the plain host language or the extended language. In contrast to model annotation no linking through URIs or similar constructs is required. New model-elements are either introduced by specialization (i.e. inheritance), connection to the extended language's model-elements, or simply by their direct insertion into an ontological level. This frees the modeler from learning about a third language to link the augmentation and base language. The multi-level modeling environment immediately recognizes the added types and offers them to the modeler. The visualization and behavior in the editor of model-elements of the extended modeling language can be completely overridden using the domain-specific language capabilities. An augmented language with a completely new look and feel can thus be created with ease by a modeler. When deciding to ignore an augmentative model when viewing a modeling language instance, all augmentation model-elements and information derived from them should be hidden. In such a case, the supertypes of the augmentation model-elements, defined in the original language, should be used to visualize instances.

Fig. 7 shows how a multi-level modeling environment, extended with such dynamically importable augmentation models (indicated by a grey background), can be used to support language augmentation in the context of our running example. For convenience, we use the UML package notation to represent the BPMN and MDPE augmentation models. Both models are stored remotely which is indicated by the “@” character followed by a URI underneath the model's name. The BPMN model is loaded from http://example.org/bpmn and the MDPE model from http://example.org/mdpe. Local models are indicated by leaving out the location information. The generalization between remote models is stored in the local ontological level, which in this case is O_0. It is important that statements about the relationships between remote models do not belong to the remote models themselves so that they can remain decoupled from one another and can exist independently. Transformations defined on the content of a model can be reused after wiring models together because the model-elements on which the transformation is defined continue to exist. For example, in Fig. 7, a transformation defined on the BPMN model which transforms a business process to a work flow engine format would still work after including the model-elements from the MDPE augmentation model. These inherit from the types defined in the BPMN model on which the transformation was defined on and do not alter them.

The combination of different modeling languages into one modeling language is a scenario that happens quite often. Examples for these are the development of computer based systems [37] where hardware design, software

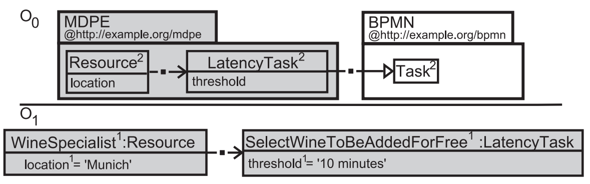


Fig. 7. A BPMN package is augmented with LatencyTask and Resource.

engineering, performance modeling and others are combined or the combination of business process modeling, simulation and software modeling [23]. The example in Fig. 8 shows a business process modeling language which is augmented with information from two different domains (indicated by grey background) at the same time—performance information from the MDPE augmentation model and security information from the Security augmentation model. Many more additional augmentation models can be added to the same base language to extend it with concepts from different domains. The augmentation models modify the concrete syntax of the DSL so that all latency-related domain-specific symbols have a L in their upper right corner to indicate that they are special in terms of their relation to the performance augmentation. SecurityTasks have an S in the upper right corner to indicate that they are security related.

To easily work with a model containing so much information, a modeler can reduce complexity by generating different views on the model. The arrows — (1), (2), (3), (4)—show the different views on the model that can be generated on-the-fly by the tool. The first view of the model that can be viewed is (1) containing the performance and business process model. This enables a modeler to view only the business process performance information with unimportant facts for this perspective filtered out. The bottom of (1) shows the corresponding rendering of a LatencyTask. It overrides the concrete default syntax of the business process modeling language's Task construct to indicate that a LatencyTask is shown. This is achieved by displaying a small L in the upper right corner of the Task. Additionally, the defined threshold for the execution time is displayed in the upper right corner of the LatencyTask. The next view (2), which at first sight is the most obvious one, shows the model with all information included. Such a view can become very complex due to the mix of different augmentations from different problem domains, but it can also provide a unique opportunity to analyze a business process combining information from different problem domains using the general-purpose and domain-specific notations. To resolve a possible rendering conflict, the closest common supertype of the classification hierarchy is used to render the clabject which in this case is Task. For pure process modeling a modeler can project the view indicated by (3), which contains just the business process information. In this view the model complexity is reduced and focused on process modeling. As with the performance view, a modeler can create a pure security view on the business process as indicated by (4). The bottom of this view consists of a SecurityTask displayed via a custom visualization. The S in the upper right of the figure indicates that a SecurityTask is shown. The 200 in the upper left of the Task symbol indicates that a business value of 200 has been added to the process input by processing it through this Task. All of these views show different aspects of the overall model, each incorporating information from a different augmentation model. Having all these different options available allows a domain expert to view a diagram using the language that is optimized for his role and experience. This frees him from the clutter of information that is unnecessary for the task in hand.

Strong tool support is necessary to handle the earlier presented scenario including distributed models and different views. The in [38] formalized classification and validation rules are encoded in our multi-level modeling language workbench. These rules guarantee valid models created by a user when using the tool. Additionally, strong

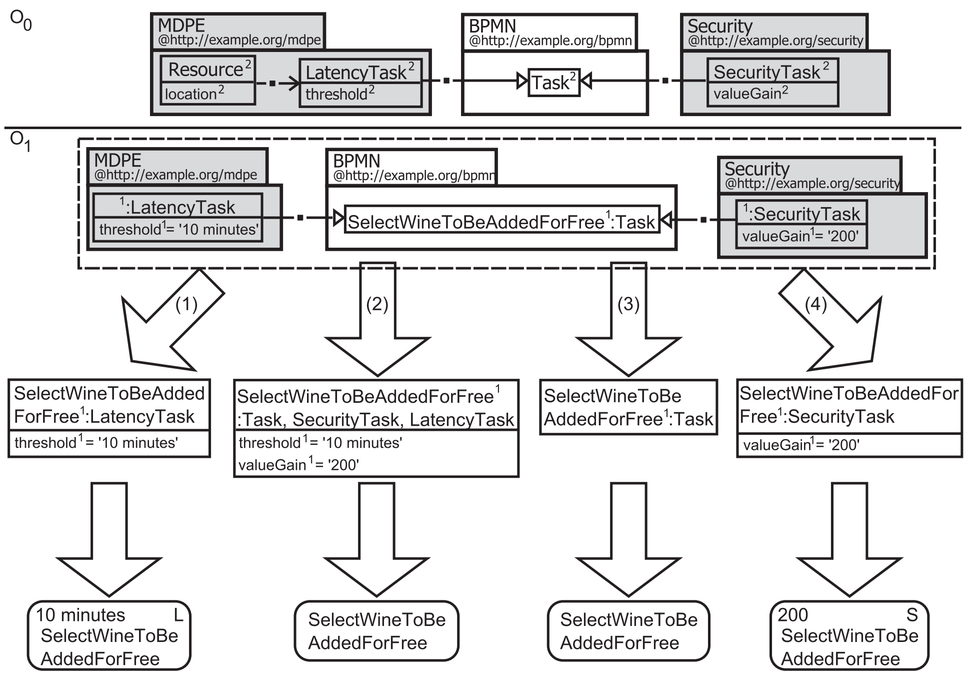

O0
MDPE @http://example.org/mdpe
Resource2 location2
LatencyTask2 threshold2
BPMN @http://example.org/bpmn
Task2
Security @http://example.org/security
SecurityTask2 valueGain2
O1
MDPE @http://example.org/mdpe
1:LatencyTask threshold1= '10 minutes'
BPMN @http://example.org/bpmn
SelectWineToBeAddedForFree1:Task
Security @http://example.org/security
1:SecurityTask valueGain1= '200'
(1)
(2)
(3)
(4)
SelectWineToBeAddedForFree1:LatencyTask threshold1= '10 minutes'
SelectWineToBeAddedForFree1:Task, SecurityTask, LatencyTask threshold1= '10 minutes' valueGain1= '200'
SelectWineToBeAddedForFree1:Task
SelectWineToBeAddedForFree1:SecurityTask valueGain1= '200'
10 minutes L SelectWineToBeAddedForFree
SelectWineToBeAddedForFree
SelectWineToBeAddedForFree
200 S SelectWineToBeAddedForFree

Fig. 8. A business process core language with a performance and security annotation packages.

tooling support is necessary to make statements about model-elements which do not fit in a single view on the model in such a distributed modeling scenario. Hence, a strong designation mechanism is needed to make statements about information transported by model-elements which are not visible at a certain point in time. The designation approach described in [36] is available in our tooling and allows to make statements about the inheritance, classification and location hierarchy of a model-element in its name compartment. We are planning to further extend this mechanism to for instance designate the attributes and methods compartments of a model-element. At the moment we are working on mechanisms to efficiently handle compositions of distributed models. This mechanism shall allow linking to remote models and transporting the evolution of remote models to local models and their instances. At present out tool supports such an evolution mechanism which helps keeping our formalized classification rules valid during evaluation for local models only. This support, however, can be extended to work over a network using change recording and application to local models. This is a problem which often occurs in the domain of ontology modeling and is for example solved in [39]. Graphical DSL view generation including elision options is also needed. Our tool has currently a visualization search algorithm included which allows to view the default general purpose and one user-defined domain-specific notation. The algorithm however can be extended to also view different domain-specific notations as needed. Elision is about hiding model-elements and making statements about these hidden elements as needed. The tool does support elision in different forms at the moment but we are planning a major refinement of this feature. The here listed extensions which are not covered by the tool yet are minor implementation work and in most cases extensions to existing implementations. Thus, we are very confident that we can upgrade our tooling to fully support all of the necessary features to support the full distributed language augmentation and enhancement scenario in the near future.

The next section shows how the augmentation techniques presented in this paper can be utilized in a fictive company selling bottles of wine to optimize its shipment process.

6. Enterprise scenario: analyzing the wine shipping process

After modeling all of their business processes, a company decides to augment their business process with information to measure their performance. Furthermore, the company is concerned about the security of certain tasks in the business process. The company has no expertise and tooling in place for both domains. Hence, a solution which offers reuse of domain knowledge, external consultants familiar with the languages and connection to existing tools is desired. This leads to the decision to reuse existing standardized language augmentations with connection to tools. For both domains, the security and performance augmentation models fulfilling these requirements exist. The company decides to use the MDPE augmentation model available at http://example.org/mdpe and the Security augmentation model available at http://example.org/security. After loading the augmentation language, the features are made available to a modeler by inheriting from the Task construct of the process modeling language. This offers the option to model performance related LatencyTasks which allow to define a maximum threshold for the execution time of a task and a SecurityTask defining a valueGain to raise awareness of Tasks which are especially important to the company. An excerpt of the augmented business process modeling language is shown in Fig. 8 at level O_0.

The most important process in the company is the process of shipping wine bottles. Hence, a security expert is contracted to secure this process. The SelectWineToBeAddedForFree task is identified as the most important one. Customers with a high revenue get free bottles of wine to raise their loyalty, whereas giving free bottles of wine to customers with a low revenue creates additional costs without any benefits. Also the choice of the free bottle of wine plays an important role to satisfy customers. In the past the importance of this task was underestimated by the employees and thus executed in an ad hoc way. To raise awareness of the task's importance a valueGain which is significantly higher than the value of all other tasks is defined for it. The view on the task from the security consultant's point of view is shown in the bottom right of Fig. 8, (4).

After a few months, the company notices that more and more customers start complaining about very high shipping times. To find the reason for the complaints a performance expert is contracted to perform a performance analysis of the shipping process. The business performance consultant calculates the maximum thresholds for each task in the process. The value calculated for SelectWineToBeAddedForFree is 10 minutes. The view of the performance consultant is shown in the bottom left of Fig. 8, (1). The performance analysis shows that the business process is dramatically missing its execution threshold because of the SelectWineToBeAddedForFree task.

The results of the performance analysis are reported to the management. To find the reason for the business process delay, the SelectWineToBeAddedForFree task is analyzed using all information available. Thus, a view displaying all information about the task is created. Such a view is shown in the middle left at the bottom of Fig. 8, (2). This view showing the model element with all augmentation information makes it possible for a unique holistic analysis of the business process to be performed. It uncovers the fact that the execution target of 10 minutes is totally incompatible with the defined value gain of 200. To solve the problem without hiring additional employees it is decided to raise the execution threshold to 20 minutes and lower the value gain so that employees spend less time on the SelectWineToBeAddedForFree task.

The scenario highlights several advantages of the multi-level augmentation features. The company could enrich its business processes with existing languages for the definition of business process performance and security. This saved time and cost for creating languages in a domain the company is not an expert in. Missing domain knowledge could be bought from consultants who are experts in the domains and the used languages. Existing tools connected with the language augmentations can be used after augmenting the business process. The view generation allowed each of the experts to use a business process notation which best fitted their needs. The availability of a holistic view made it possible to effectively solve the problem with the business process.

After illustrating the capabilities of the multi-level augmentation mechanism on a fictitious enterprise scenario in the next, section we evaluate its capabilities in a real-world scenario to show the practical usability of the approach.

7. Evaluation

Multi-level modeling approaches have been used in one form or another in a variety of industrial and standardization projects. The improvements suggested in this paper represent an incremental extension of these approaches, albeit a very important one. The utility of the overall multi-level modeling approach has therefore to a large extent been demonstrated by the success of these projects as reported in the papers cited in the following related work section. This evaluation focuses on presenting the benefits of the enhancements suggested in this paper in the context of a high-profile application of the approach. The scenario used to do this is based on Gonzales–Peres and Henderson–Sellers's work on using powertypes and clabjects in the ISO/IEC24744 standard "Metamodel for Development Methodologies" in [40] whose goal is to support the modeling of different development activities, primarily within software engineering methods. This standard of which an exemplary application is shown in Fig. 9 is composed of three layers — a fixed Metamodel together with the Metamodel extension layer describing the general kind of activities available, the Method layer providing a method specialist with the ability to introduce concepts needed in a particular scenario and the Endeavor layer which is used by the person executing the defined methodology.

The layers are indicated by a gray background in Fig. 9. This excerpt shows the Task related part of the modeling language. A powertype pattern [42] is used to allow a user to introduce different kinds of tasks. The powertype is TaskKind which is partitioned by Task using Name as discriminant. The subclasses of Task (e.g. PackageCDsTask) are instances of TaskKind. Using this approach it is possible to simulate three classification levels in the UML despite the fact that the language only supports two levels directly — Classes and Object Specifications. The instances of the powertype represent the instance facet and the subclasses of the partioned type represent the type faced of the model-element. Combining these two facets into one concept gives rise to the notion of clabjects as earlier introduced. In [40], clabjects are represented informally using a dashed ellipse to encapsulate the type and instance components of a clabject as shown in Fig. 9 since the two facets can only be represented separately in the UML. Using this powertype pattern a method engineer can extend a method with Tasks that are needed in specific methodologies. Here a PackageCDsTask is introduced into the methodology. Another subclass of Task that could also be envisaged is "ValidateRequirementsTask" which carries a "NeedsSignOff" attribute stating that some of these tasks need to be signed off. The powertype example in Fig. 9 not only facilitates the introduction of different tasks at the Method layer but also the introduction of new concepts from different domains. The example extends the meta-model with new language constructs from the performance domain which allow performance related tasks to be modeled next to Tasks. For this reason a new powertype, MeasuredTaskKind, and partitioning type MeasuredTask are introduced into the Metamodel layer. In the example, the powertype is

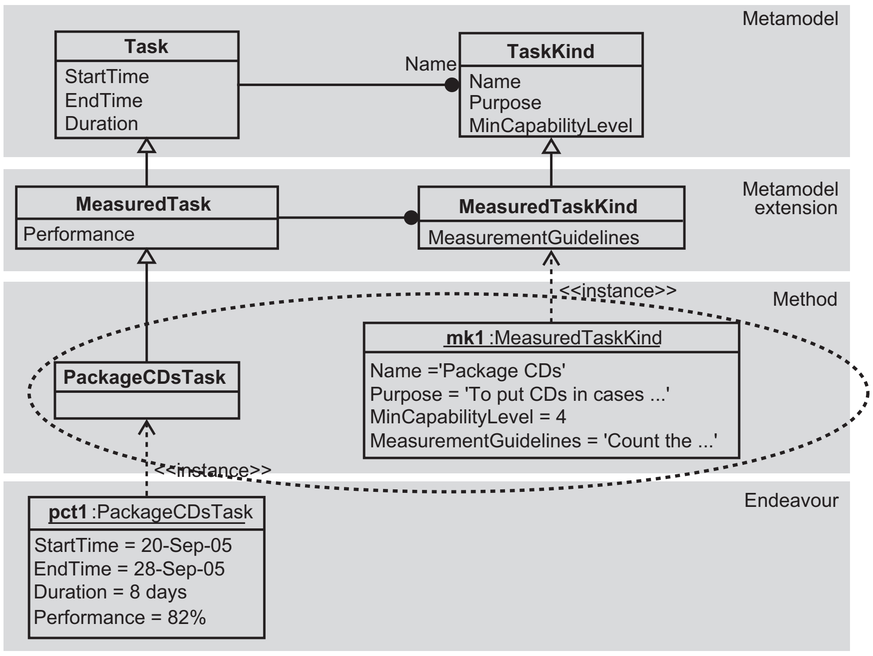


Fig. 9. Example after [40,41].

instantiated by a PackageCDsTask at the Method layer. The type facet of PackageCDsTask is represented by the object specification mk1:MeasuredTaskKind which specifies the values of the type facet as shown in Fig. 9. It can be observed that in pct1:PackageCDsTask the values specified in mk1 are not available anymore. The PackageCDsTask is instantiated at the third classification level, the Endeavour layer. The instantiation is represented through an object specification (pct1: PackageCDsTask) which started at 20-Sep-05 and ended at 28-Sep-05 with a performance of 82%.

7.1. *Translation to multi-level modeling*

Although this approach shown in Fig. 9 fulfills the basic goal of allowing a modeling language to be enhanced and augmented it has some disadvantages. Firstly, it uses an advanced feature (powertypes) that is not universally supported in all UML tools. Secondly, the resulting models are more complicated than they need to be because of the lack of a language feature for directly representing clabjects as a single modeling element. In a sense, the approach simulates multi-level modeling using the limited modeling features available in the UML. In the following we show how the example can be modeled in a simpler and more expressive way using a notation that fully supports the multi-level modeling approach.

To compare the powertype approach to the full multi-level modeling approach using this ISO standard scenario we first translate the powertype pattern to its equivalent one-to-one multi-level modeling representation using potency-based clabjects. This requires that all model-elements are placed at the classification level that corresponds to their classification level in the original version. To translate a powertype pattern into a multi-level model the following algorithm is applied: (a) Move the powertype the first ontological; (b) the partitioning type is moved to the second level; (c) the subclasses of the partitioning type are moved to the same level and are declared as subclasses of it; (d) the subclasses of the partitioning type are declared as instances of the powertype; (e) the values given to object specifications of the powertype are used as default values for the subclasses of the partitioning type. Fig. 10 shows a translation of the example in Fig. 9 to an equivalent multi-level model. The figure shows the powertype example resolved to three classification levels. Furthermore classification is clearly expressed through a classification notation and is not composed of two connected model-elements of which the subtypes of one are regarded as instances of the other. It can be observed that powertypes (TaskKind, MeasuredTaskKind) and the partitioned type (Task, MeasuredTask) are mapped to two separate levels. This reflects the fact that the subtypes of Task and MeasuredTask are instances of TaskKind and MeasuredTaskKind respectively which would lead to a generalization relationship crossing borders if the powertype and the partitioned type were placed on the same classification level. It can be noted that Task no longer needs an association to TaskKind to indicate what attribute is used for partitioning. The same is true for MeasuredTask and MeasuredTaskKind. PackageCDsTask is placed at the same classification level as MeasuredTask which it is a subclass of. Furthermore, it is an instance of MeasuredTaskKind as in the

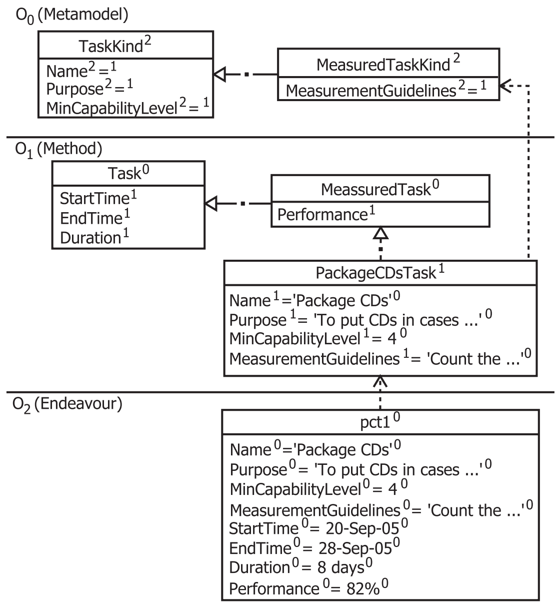


Fig. 10. Translation to multi-level modeling of the example from [40].

powertype example. On the lowest classification level pct1 is defined as an instance of PackageCDsTask.

The potencies of the clabjects in this new version provide a way of controlling over how many levels a clabject can be instantiated, an attribute can be handed over to instances and a value changed. The figure displays the Metamodel clabjects with a potency of 2 stating that these can also be instantiated on the Method and Endeavour level. Their attributes have the same potency (i.e. durability) since they should be available on these two classification levels. The potency of the values (i.e. mutability) is set to 1 indicating that they can only be changed by the method engineer and not at the Endeavour level. The clabjects Task and MeasuredTask, residing at the Method level, are defined to be abstract by setting their potency to 0. Their attributes, in contrast, have a durability of 1 since they should be handed over to instances of subclabjects. Their mutability is also 1 as they are intended to be changed at the Endeavour level. The concrete task PackageCDsTask is instantiated from MeasuredTaskKind and thus has a one-lower potency of 1. Since its attributes have a durability of 1 and mutability of 0 they can be handed over to instances but their values are fixed after defining them in the Method layer. The Endeavour layer contains only elements with a potency of 0 as it is the most concrete layer and cannot be instantiated further. The extension scenarios described in [40] are still available in this new version. New attributes can be introduced by adding them to the clabjects at the Method layer and new meta-model concepts can be introduced by using inheritance at the Metamodel layer.

7.2. *Refinement of the multi-level model*

Using the full power of the multi-level modeling approach the example from Fig. 9 which is translated to deep-modeling in Fig. 10 can be further refined to reduce accidental complexity introduced by applying the powertype pattern. Although the example in Fig. 10 uses multi-level modeling features to model the MeasuredTask scenario, it does so in the style of Fig. 9. In other words, it retains all the conceptual model-elements in the original version, but maps them to a unified clabject notation. The resulting model is a perfectly valid multi-level model, but does not fully leverage multi-level modeling's ability to present durability and mutability information in a compact way. Nor does it demonstrate the multi level-aware model composition mechanism proposed in this paper.

Fig. 11 shows a refined multi-level model containing the same information as the original powertype version in Fig. 9, but using the full power of multi-level modeling to represent the information in a minimalistic way. The figure also shows how the multi level-aware model composition mechanism can be used to organize enhancements and augmentations in conceptually coherent groups. The main difference to Fig. 10 is that the partitioning types have been removed and their attributes have been moved to the Metamodel level. This frees the method engineer from having to use inheritance relationships in the Method layer. The superclasses can, however, be introduced again if the method engineer wanted to show the partitioned classes. This is done by introducing a common superclass for these elements in the Method layer. Set theoretic statements about the subclasses can also be made (e.g. whether or not the specialization of Tasks in a method is complete).

The minimalistic multi-level version in Fig. 11 makes heavier use of mutability values for attributes since many values are fixed from the Method level on (the attributes originating from the powertypes) and others are changeable even at the Endeavour domain (the attributes originating from the partitioned type). The new model has about half the number of model-elements as the original one (4 elements in comparison to 7), thereby reducing the complexity of the model. Other advanced features supported by multi-level modeling tools, but not described in this paper, can also be applied to the new model such as the application of different graphical and textual visualizations on-the-fly at different classification levels, validation of classification relationships or the distribution of extensions and models over the internet including remote model emendation.

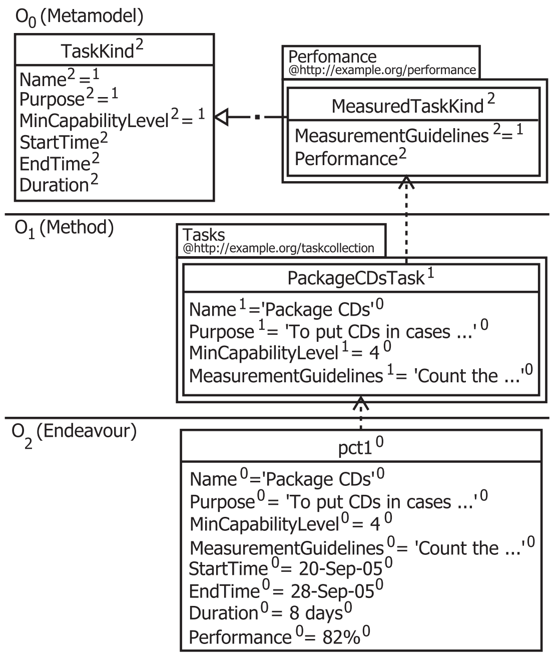
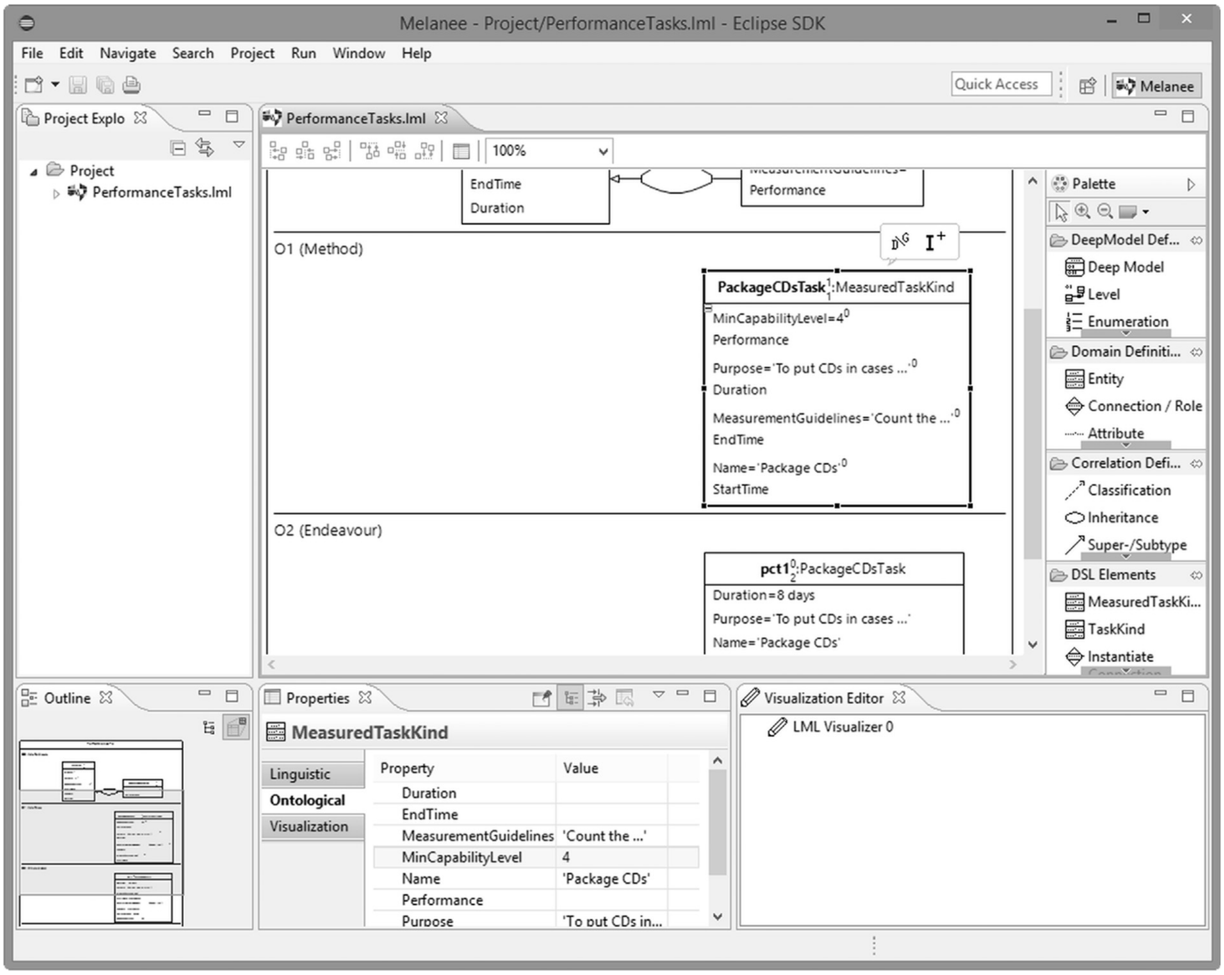


Fig. 11. Optimized multi-level model of the example from [40].

Finally, the example shows how level-aware model composition can be used to group related extension concepts so that they can be reused wherever needed simply by composing models. The Performance augmentation model at the Metamodel level contains all the performance related extensions of TaskKind that can be reused in different methods as required (in this example there is only one extension, but in general there would be numerous). The Tasks model at the Method level shows an example of a specific methodology using the extensions in Performance by instantiation.

7.3. Implementation in a multi-level modeling tool

An implementation of this example in the Melanee tool is shown in Fig. 12. The left of the screenshot shows the project and file management capabilities of the tool allowing diagrams to be sorted into projects which can contain folders. The lower left shows a graphical outline of the currently opened model. This view can be used to navigate quickly over large multi-level models. The diagram editor in the middle shows the example discussed in this section. The Method layer is at the focal point of the editor and the PackageCDsTask clabject is selected. A popup menu at the top of the selected clabject shows the options which can be applied to it. Here, the option to toggle between a DSL rendering and GPL rendering is displayed at the left and the option to instantiate the clabject is displayed on the right of the popup. The right side of the editor displays the palette which offers different model-elements for creation. The folder at the bottom of this palette, the DSLElements folder, offers elements for instantiation in a context sensitive way depending on what element is selected in the editor. In this screenshot, MeasuredTaskKind and TaskKind are offered for instantiation because a model-element at the Method layer is selected. The bottom left of the screenshot shows a Properties editor. This editor allows linguistic properties to be manipulated in the Linguistic tab and ontological properties in the Ontological tab. Here, the Ontological tab is selected displaying the ontological attributes of the PackageCDsTask clabject. The lower right of the screenshot shows the Visualization Editor. This editor allows the general purpose rendering of a clabject to be changed (i.e. by forcing certain things to be shown or hidden), adding a graphical domain-specific language visualization to a clabject or adding a textual domain-specific language visualization to a clabject. We are currently working to allow visualization of validation information and plan to provide other visualizations such as tables and wikis in the future. It is also possible to customize the modeling environment itself through visualizers in a multi-level model so that a modeler gets an environment that is optimized for his needs. We are also



Property	Value
Duration	
EndTime	
MeasurementGuidelines	'Count the ...'
MinCapabilityLevel	4
Name	'Package CDs'
Performance	
Purpose	'To put CDs in...

Fig. 12. Optimized multi-level model of the example from [40] in Melanee.

refactoring the meta-model of Melanee to cleanly support models which are distributed over the internet and natively storable in different formats such as graph-based databases, RDF and XMI (which is currently the only supported format).

7.4. Discussion

The example has demonstrated the applicability of the multi-level modeling approach to a significant real world scenario, an ISO standard, which has a strong need for meta-model extensions. Apart from the proposed level-aware packaging mechanism we only used the core capabilities of multi-level modeling which focus on modeling multiple classification levels and the introduction of types at different levels of classification on-the-fly. Thus all modeling tools implementing multi-level modeling using the same linguistic meta-model can support and exchange models in the presented modeling domain. A useful next step would be to establish a so called multi-level model standard, similar to the UML standard, which could then be supported by all multi-level modeling tools.

In the second version of the example (Fig. 11) we were able to show that multiple classification levels can be more clearly and concisely expressed using the full multi-level modeling approach. The multi-level representation using clabjects not only expresses the scenario using fewer model elements, it also minimizes the scattering of related and dependent information over different parts of the model. Moreover, multi-level modeling allows classification information to be expressed by classification relationships rather than through connections and subclasses which need to be interpreted as classification relationship relations by a modeler. Moreover, in multi-level modeling tools the semantics of classification is supported and enforced across all classification levels, whereas it is not clear if all UML tools which are used to model powertypes are capable of providing classification validation support across multiple classification levels.

Another advantage of the multi-level modeling versions of the model is that a clear syntax for model-elements representing the class/object duality (i.e. clabjects) is available instead of the artificial, non-standard concrete syntax of an ellipse surrounding the type and instance facets of a model-element. By reducing the number of model-elements to almost half the original number, the new version of the model significantly reduces accidental complexity. Moreover, various potency concepts in the deep-instantiation mechanism allow the durability and mutability of attributes and their values to be more precisely expressed. In the multi-level modeling version a modeler can explicitly express in one place over how many levels an attribute endures or can be changed whereas in the powertype version this information is scattered over the instance and type facets of the types at the Method level. Moreover, in the powertype example, this only works as long as attributes of the instance facet should not endure to instances of the type facet or if a value shall not be changed by instances of the type facet. If attributes are mandatory for the type and instance facet in the powertype approach they would have to be duplicated and kept synchronized by hand as values at the instance facet play the role of default values in the type facet. In such a case the question arises how to handle values which need to be handed over to instances but be constant from the type facet onwards. Such a scenario can only be supported in the UML using a naming convention or OCL constraint since an explicit modeling construct is not available in the language itself. Multi-level modeling addresses this problem by supporting a level-agnostic representation of clabjects that explicitly embraces the type/instance duality of model-elements. The need for a non-standard syntax is therefore obviated. In multi-level modeling, the durability of attributes (i.e. whether they are handed over to instances) and the mutability of attribute values (i.e. whether they can be changed in an instance of a clabject) can be explicitly defined using the built-in potency and mutability mechanisms.

8. Related work

Although it is not the only motivation for multi-level modeling, the provision of simpler and more flexible extension mechanisms has always been one of the main ones [43–45]. Moreover, the importance of supporting flexible language extension mechanisms in business process modeling and other areas of business informatics has been highlighted in studies such as [46]. Over the last few years there has been a big increase in the number of approaches and tools claiming to support multi-level modeling and in the number of groups that have used the approach in real-life scenarios. For example, the Resourcebus infrastructure of the Metada startup company is explicitly advertised as supporting multi-level modeling based on the orthogonal classification architecture. Another tool that is used commercially in domains such as finance, aerospace and telecommunications [47] is XModeler [48].

Besides these industrial tools multi-level modeling has also been applied to improve standards and methodologies. It is for example applied to ISO 15926 in [49]. This work focuses on making a flat model a deep model, allowing types to be introduced at one level and instantiated at another level (e.g. the creation of a new type of pump by an engineer at level O_1 and input of maintenance data for a specific pump instance at level O_2). The authors conclude that the complexity of extended and extensible models is dramatically reduced by applying multiple levels over the previously used flat modeling formalism. In [50] multi-level modeling was applied to industrial automation systems which consist of hundreds of model-elements. Their evaluation showed that multi-level modeling is applicable for real world scenarios containing big models. In [51] multi-level modeling is applied to a real world software product line engineering scenario. In [52] multi-level modeling was used to create a meta-language called QML/CS [53] for modeling quality-of-service-constraints.

Lara and Guerra describe a textual, multi-level modeling tool named Meta-Depth [54,55]. Although the tool is less suitable than graphical deep-modeling tools for communicating high-level structures in a business context it does provide a rich transformation, query and action language. For this purpose the Eclipse Epsilon Framework [56] has been tailored to support a multi-level modeling framework. The Epsilon framework is very closely oriented towards OMG standards like HUTN [57] and OCL [58] making Meta-depth a good choice for users wanting to express textual models and leverage a powerful transformation/query/action language support. MetaDepth also allows models to be distributed across files. Another approach that supports textual deep-model content is the work on DeepJava [59].

In [60,61] Volz et al. give an overview of their Open Meta Modeling Environment (OMME). The tool allows graphical DSLs to be created comparable to the Melanee tool used in this work. In contrast to the Melanee tooling presented here the developers of OMME plan to support the freestyle creation of models without semantics. Such a model can then later be connected to a meta-model instance. A modeler for example can simply draw a model based on a standard library of shapes and can then subsequently define a mapping between the graphical model and a model carrying semantics (e.g. instances of an entity relationship model). To create new DSLs the tool allows predefined figures to be assembled into a new graphical metaphor. If a user does not find a suitable shape a figure programmed in Java can be provided. The meta-model extension and graphical DSL definition scenario presented in this paper can also be achieved using the OMME approach.

Henderson-Sellers et al. [47] describe an architecture for deep-modeling in which everything is an object. This architecture is used to implement the formerly mentioned XModeler tool [48]. Creating domain specific languages is supported by implementing clients provided by XModeler. The four main clients are Diagrams, Forms, ModelBrowser and TextEditor.

Cross-layer modeler [62] is a tool reassembling multiple UML notations and views to create models at different levels and describe their classification relationships. A constraint language is supported to define constraints for model-elements across multiple levels. A unique feature of this tool is the ability to allow constraints to be automatically refactored when changes are made to a model.

Nivel [63], is a deep-modeling language with a well defined formal semantics. The semantics is defined using the weight constraint rule language (WCRL) [64], which can also be used to define constraints on a model. Unfortunately, according to [63], no Nivel tool is available yet. The DPF workbench [65] is a graphical multi-level editor which provides a graph based formalization of its underlying modeling language. It has support for creating graphical domain-specific editors and allows constraints to be defined on models. Code can be generated from DPF models by a code generator.

A tool which is not deep-modeling related but also supports meta-model adaption is MetaEdit+ [66]. This tooling, however, does not support scenarios spanning multiple classification levels as shown in this work. The editors for defining graphical languages are the most mature ones known to the authors at the time of writing. A drawback of MetaEdit+ is its use of a proprietary format which can be worked around by leveraging its interoperability features provided through its API and XML connectivity component. Other tools supporting the creation of domain-specific languages in a two-level scenario are the Generic Modeling Environment [67], GMF [68] and Sirius [69]. The latter two are official Eclipse projects and thus wide spread. AToMPM [70] the successor of AToM^3 [71] allows to model using different graphical notations for one model.

9. Conclusion

In this paper we examine the different modeling language extension strategies currently supported in mainstream modeling environments in the domain of enterprise systems, and evaluate their suitability for supporting two key language extension scenarios – language enhancement (where the extensions are in the same domain as the core language) and language augmentation (where they are not). Each of the three identified strategies – meta-model customization (in which the language meta-model is directly enhanced), the use of built-in extension mechanisms (in which meta-model customization is simulated but the original hardwired meta-model is kept unchanged) and model annotation (in which instances of the original meta-model are connected to instances of extension concepts using model weaving) have different strengths and weaknesses from the perspective of these two scenarios (enhancement and augmentation). The chief weaknesses are the need for tool recompilation and redeployment in the case of meta-model customization, the lack of uniform support and accidental complexity in the case of built-in extension mechanisms, and the introduction of redundancy and significant overhead in the case of model annotation. At the present time it is not possible for modelers to enhance a modeling language without suffering from one of these weaknesses.

Multi-level modeling infrastructures based on the OCA have the potential to address this problem at a fundamental level and provide support for language extensions free from any of these weaknesses. In particular, their separation of ontological and linguistic classification levels into two separate dimensions effectively unifies the (meta)-model customization and built-in extension mechanism approaches, allowing model enhancement to be achieved without any of the identified weaknesses. In this paper, however, we also observed that the current generation of multi-level modeling approaches have a key shortcoming when it comes to supporting language augmentation – the lack of a built-in model composition mechanism capable of supporting the dynamic loading of different augmentations. The incorporation of such a mechanism, coupled with the flexible domain-specific visualization mechanism not only addresses this shortcoming it also allows modelers to switch language augmentations on and off, at will, during the modeling process. The support for the definition of symbiotic domain-specific and general-purpose modeling languages also allows modelers to dynamically select whether model-elements are visualized using a domain-specific or general purpose syntax.

Based on the insights gained in this paper we are currently upgrading Melanee to support the presented dynamically selectable model composition mechanism, and thereby offer a modeling environment free of any of the weaknesses identified in the paper. We will report on our progress in future papers. While there will always be a need for cross-platform model annotation solutions of the kind outlined in Section 2 for practical interoperability and legacy technology issues, we hope the envisaged approach will pave the way for more flexible and powerful multi-level modeling environments of the future. While the tool is useable in any domain, a major focus of our work is modeling enterprise architectures and business process applications. Once the new version of the tool is available, therefore, we plan to populate it with families of modeling languages suitable to support the domain-specific views of standards such as ArchiMate, TOGAF and other multi-view enterprise visualization standards.

Acknowledgements

We want to thank Katharina Markert for here feedback on this paper.

References

[1] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework 2.0, 2nd edition, Addison-Wesley Professional, 2009.
[2] M. Lankhorst, H. Proper, H. Jonkers, The architecture of the ArchiMate language, in: T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, R. Ukor (Eds.), Enterprise, Business-Process and Information Systems Modeling, Lecture Notes in Business Information Processing, Springer, Berlin Heidelberg, 2009, pp. 367–380.
[3] V. Haren, TOGAF Version 9.1, 10th edition, Van Haren Publishing, 2011.
[4] J.A. Zachman, A framework for information systems architecture, IBM Syst. J. 26 (3) (1987) 276–292.
[5] OMG, OMG Unified Modeling Language ™ (OMG UML), Superstructure Version 2.4.1, ⟨http://www.omg.org/spec/UML/2.4.1⟩, 2011.
[6] Object Management Group, Business Process Model and Notation (BPMN) Version 2.0, ⟨http://www.omg.org/spec/BPMN/2.0⟩, 2011.
[7] J. Bézivin, F. Jouault, P. Valduriez, First experiments with a model-weaver, in: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2004.
[8] C. Atkinson, R. Gerbig, M. Fritzsche, Modeling language extension in the enterprise systems domain, in: Enterprise Distributed Object Computing Conference (EDOC), 17th IEEE International, 2013, pp. 49–58. http://dx.doi.org/10.1109/EDOC.2013.15.
[9] F.P. Brooks Jr., No silver bullet essence and accidents of software engineering, Computer 20 (4) (1987) 10–19, http://dx.doi.org/10.1109/MC.1987.1663532.
[10] C. Atkinson, T. Kühne, Reducing accidental complexity in domain models, Softw. Syst. Model. 7 (3) (2008) 345–359, http://dx.doi.org/10.1007/s10270-007-0061-0.
[11] The Open Group, ArchiMate 2 Tool Certification Register, ⟨http://www.opengroup.org/certifications/archimate/ts-register⟩, 2013.
[12] M. Fritzsche, J. Johannes, U. Aßmann, S. Mitschke, W. Gilani, I.T.A. Spence, T.J. Brown, P. Kilpatrick, Systematic usage of embedded modelling languages in automated model transformation chains, in: SLE, 2008, pp. 134–150. http://dx.doi.org/10.1007/978-3-642-00434-6_9.
[13] M.D.D. Fabro, Metadata management using model weaving and model transformation (Ph.D. thesis), Universite de Nantes, 2007.
[14] C. Barreto, V. Bullard, T. Erl, J. Evdemon, D. Jordan, K. Kand, D. König, S. Moser, R. Stout, R. Ten-Hove, I. Trickovic, D. van der Rijn, A. Yiu, Web Services Business Process Execution Language Version 2.0,

⟨http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf⟩, 2007.

[15] A. Scheer, Business Process Engineering: Reference Models for Industrial Enterprises, Springer, 1998.

[16] M. Fritzsche, Performance related decision support for process modelling (Ph.D. thesis), Queen's University Belfast, 2010.

[17] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-oriented programming, in: M. Aksit, S. Matsuoka (Eds.), ECOOP'97 Object-Oriented Programming, Lecture Notes in Computer Science, vol. 1241, Springer, Berlin Heidelberg, 1997, pp. 220–242, http://dx.doi.org/10.1007/BFb0053381.

[18] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold, An overview of aspectJ, in: J. Knudsen (Ed.), ECOOP 2001 Object-Oriented Programming, Lecture Notes in Computer Science, vol. 2072, Springer, Berlin, Heidelberg, 2001, pp. 327–354, http://dx.doi.org/10.1007/3-540-45337-7_18.

[19] J. Whittle, P. Jayaraman, MATA: a tool for aspect-oriented modeling based on graph transformation, in: H. Giese (Ed.), Models in Software Engineering, Lecture Notes in Computer Science, vol. 5002, Springer, Berlin, Heidelberg, 2008, pp. 16–27, http://dx.doi.org/10.1007/978-3-540-69073-3_3.

[20] J. Kienzle, W. Al Abed, F. Fleurey, J.-M. Jézéquel, J. Klein, Aspect-oriented design with reusable aspect models, in: S. Katz, M. Mezini, J. Kienzle (Eds.), Transactions on Aspect-Oriented Software Development VII, Lecture Notes in Computer Science, vol. 6210, Springer, Berlin, Heidelberg, 2010, pp. 272–320, http://dx.doi.org/10.1007/978-3-642-16086-8_8.

[21] P.H. Nguyen, J. Klein, Y. Le Traon, Model-driven security with a system of aspect-oriented security design patterns, in: Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling, VAO '14, ACM, New York, NY, USA, 2014, pp. 51:51–51:54. http://dx.doi.org/10.1145/2631675.2631683.

[22] O. Alam, J. Kienzle, G. Mussbacher, Concern-oriented software design, in: A. Moreira, B. Schätz, J. Gray, A. Vallecillo, P. Clarke (Eds.), Model-Driven Engineering Languages and Systems, Lecture Notes in Computer Science, vol. 8107, Springer, Berlin, Heidelberg, 2013, pp. 604–621, http://dx.doi.org/10.1007/978-3-642-41533-3_37.

[23] R. Jung, R. Heinrich, E. Schmieders, M. Strittmatter, W. Hasselbring, A method for aspect-oriented meta-model evolution, in: Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling, VAO '14, ACM, New York, NY, USA, 2014, pp. 19:19–19:22. http://dx.doi.org/10.1145/2.2631681.

[24] S. Becker, H. Koziolek, R. Reussner, The Palladio component model for model-driven performance prediction, J. Syst. Softw. 82 (1) (2009) 3–22, http://dx.doi.org/10.1016/j.jss.2008.03.066.

[25] M. Fritzsche, M. Picht, W. Gilani, I.T.A. Spence, T.J. Brown, P. Kilpatrick, extending BPM environments of your choice with performance related decision support, in: BPM, 2009, pp. 97–112. http://dx.doi.org/10.1007/978-3-642-18023-1_9.

[26] SAP, Components & tools of SAP NetWeaver: SAP NetWeaver business process management, ⟨http://www.sap.com/platform/netweaver/components/sapnetweaverbpm/index.epx⟩ (download on 31st March 2012).

[27] A. Rodríguez, E. Fernández-Medina, M. Piattini, A BPMN extension for the modeling of security requirements in business processes, IEICE-Trans. Inf. Syst. http://dx.doi.org/10.1093/ietisy/e90-d.4.745.

[28] A. Rodríguez, E. Fernández-Medina, J. Trujillo, M. Piattini, Secure business process model specification through a UML 2.0 activity diagram profile, Decis. Support Syst. 51 (3). http://dx.doi.org/10.1016/j.dss.2011.01.018.

[29] C. Atkinson, M. Gutheil, B. Kennel, A flexible infrastructure for multilevel language engineering, IEEE Trans. Softw. Eng. 35 (6). http://dx.doi.org/10.1109/TSE.2009.31.

[30] OMG, OMG Unified Modeling Language™ (OMG UML), Infrastructure Version 2.4.1, ⟨http://www.omg.org/spec/UML/2.4.1⟩, 2011.

[31] C. Atkinson, R. Gerbig, B. Kennel, On-the-fly emendation of multi-level models, in: A. Vallecillo, J.-P. Tolvanen, E. Kindler, H. Störrle, D. Kolovos (Eds.), Modelling Foundations and Applications, Lecture Notes in Computer Science, vol. 7349, Springer, Berlin, Heidelberg, 2012, pp. 194–209, http://dx.doi.org/10.1007/978-3-642-31491-9_16.

[32] C. Atkinson, R. Gerbig, B. Kennel, Symbiotic general-purpose and domain-specific languages, in: Proceedings of the 34th International Conference on Software Engineering, ICSE 2012, ACM, 2012. http://dx.doi.org/10.1109/ICSE.2012.6227102.

[33] P.P.-S. Chen, The entity-relationship model—toward a unified view of data, ACM Trans. Database Syst. 1 (1) (1976) 9–36, http://dx.doi.org/10.1145/320434.320440.

[34] C. Atkinson, R. Gerbig, Melanie: multi-level modeling and ontology engineering environment, in: Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards, MW '12, ACM, New York, NY, USA, 2012, pp. 7:1–7:2. http://dx.doi.org/10.1145/2448076.2448083.

[35] D. Kantner, Specification and implementation of a deep OCL dialect (Master's thesis), University of Mannheim, 2014.

[36] C. Atkinson, R. Gerbig, Level-agnostic designation of model elements, in: J. Cabot, J. Rubin (Eds.), Modelling Foundations and Applications, Lecture Notes in Computer Science, vol. 8569, Springer International Publishing, 2014, pp. 18–34, http://dx.doi.org/10.1007/978-3-319-09195-2_2.

[37] A. Ledeczi, G. Nordstrom, G. Karsai, P. Volgyesi, M. Maroti, On metamodel Composition, in: Proceedings of the 2001 IEEE International Conference on Control Applications, (CCA '01), 2001, pp. 756–760. http://dx.doi.org/10.1109/CCA.2001.973959.

[38] B. Kennel, A unified framework for multi-level modeling (Ph.D. thesis), University of Mannheim, 2012.

[39] M. Klein, N.F. Noy, A component-framework for ontology evolution, in: Proceedings of the Workshop on Ontologies and Distributed Systems, IJCAI '03, 2003.

[40] C. Gonzalez-Perez, B. Henderson-Sellers, Metamodelling for Software Engineering, Wiley Publishing, 2008.

[41] B. Henderson-Sellers, C. Gonzalez-Perez, On the ease of extending a powertype-based methodology metamodel, in: Meta-Modelling and Ontologies, Proceedings of the 2nd Workshop on Meta-Modelling, WoMM 2006, 2006, pp. 11–25.

[42] J. Odell, Power types, J. Object Oriented Programm. 7 (2) (1994) 8–12.

[43] C. Atkinson, T. Kühne, Strict Profiles: Why and how, in: A. Evans, S. Kent, B. Selic (Eds.), UML 2000–The Unified Modeling Language, Lecture Notes in Computer Science, vol. 1939, Springer, Berlin, Heidelberg, 2000, pp. 309–322, http://dx.doi.org/10.1007/3-540-40011-7_22.

[44] C. Atkinson, T. Kühne, B. Henderson-Sellers, To meta or not to meta—that is the question, J. Object Oriented Programm. 13 (8) (2000) 32–35.

[45] D. D'Souza, A. Sane, A. Birchenough, First-class extensibility for UML—packaging of profiles, in: R. France, B. Rumpe (Eds.), UML'99 – The Unified Modeling Language, Lecture Notes in Computer Science, vol. 1723, Springer, Berlin, Heidelberg, 1999, pp. 265–277, http://dx.doi.org/10.1007/3-540-46852-8_19.

[46] D. van der Linden, H.A. Proper, On the accommodation of conceptual Distinctions in Conceptual Modeling Languages, in: Proceedings of the conference Modellierung 2014, 2014, pp. 17–32.

[47] B. Henderson-Sellers, T. Clark, C. Gonzalez-Perez, On the search for a level-agnostic modelling language, in: C. Salinesi, M. Norrie, S. Pastor (Eds.), Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 7908, Springer, Berlin, Heidelberg, 2013, pp. 240–255, http://dx.doi.org/10.1007/978-3-642-38709-8_16.

[48] T. Clark, J. Willans, Software language engineering with XMF and XModeler, formal and practical aspects of domain specific languages: recent developments. http://dx.doi.org/10.4018/978-1-4666-2092-6.ch011.

[49] A. Jordan, G. Grossmann, W. Mayer, M. Selway, M. Stumptner, On the application of software modelling principles on ISO 15926, in: Proceedings of the Modelling of the Physical World Workshop, MOTPW '12, ACM, New York, NY, USA, 2012.

[50] T. Aschauer, G. Dauenhauer, W. Pree, Representation and traversal of large clabject models, in: A. Schürr, B. Selic (Eds.), Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, Springer, Berlin Heidelberg, http://dx.doi.org/10.1007/978-3-642-04425-0_3.

[51] G. Dauenhauer, T. Aschauer, W. Pree, Variability in automation system models, in: S. Edwards, G. Kulczycki (Eds.), Formal Foundations of Reuse and Domain Engineering, Lecture Notes in Computer Science Springer, Berlin, Heidelberg, 2009, pp. 116–125, http://dx.doi.org/10.1007/978-3-642-04211-9_12.

[52] A. Alreshidi, S. Zschaler, P. McBurney, Towards a quality modelling language for component-based systems (QML/CS), in: Proceedings of the 2013 International Workshop on Quality Assurance for Service-based Applications, QASBA 2013, ACM, New York, NY, USA, 2013. http://dx.doi.org/10.1145/2489300.2489332.

[53] S. Zschaler, Formal specification of non-functional properties of component-based software systems, Softw. Syst. Model. 9 (2) (2010) 161–201, http://dx.doi.org/10.1007/s10270-009-0115-6.

[54] J. de Lara, E. Guerra, Domain-specific textual meta-modelling languages for model driven engineering, in: A. Vallecillo, J.-P. Tolvanen, E. Kindler, H. Störrle, D. Kolovos (Eds.), Modelling Foundations and Applications, Lecture Notes in Computer Science, vol. 7349, Springer, Berlin/Heidelberg, 2012, pp. 259–274.

[55] J. de Lara, E. Guerra, Deep meta-modelling with METADEPTH, in: Proceedings of the 48th International Conference on Objects, Models, Components, Patterns, TOOLS'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 1–20.

[56] D.S. Kolovos, R.F. Paige, F.A.C. Polack, Eclipse development tools for epsilon, in: In Eclipse Summit Europe, Eclipse Modeling Symposium, 2006.
[57] OMG, Human-usable textual notation (HUTN) specification, version 1.0, ⟨http://www.omg.org/spec/HUTN/1.0/⟩, 2012.
[58] OMG, OMG object constraint language (OCL), version 2.3.1 without change bars, ⟨http://www.omg.org/spec/OCL/2.3.1/⟩, 2012.
[59] T. Kühne, D. Schreiber, Can programming be liberated from the two-level style: multi-level programming with Deepjava, in: Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications, OOPSLA '07, ACM, New York, NY, USA, 2007, pp. 229–244. http://dx.doi.org/10.1145/1297027.1297044.
[60] B. Volz, S. Jablonski, Towards an open meta modeling environment, in: Proceedings of the 10th Workshop on Domain-Specific Modeling, DSM '10, ACM, New York, NY, USA, 2010, pp. 17:1–17:6.
[61] B. Volz, S. Zeising, S. Jablonski, The open meta modeling environment, in: ICSE 2011 Workshop on Flexible Modeling Tools (FlexiTools 2011), 2011.
[62] A. Demuth, R.E. Lopez-Herrejon, A. Egyed, Cross-layer modeler: a tool for flexible multilevel modeling with consistency checking, in: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, ACM, New York, NY, USA, 2011, pp. 452–455. http://dx.doi.org/10.1145/2025113.2025189.
[63] T. Asikainen, T. Männistö, Nivel: a metamodelling language with a formal semantics, Softw. Syst. Model. 8 (4) (2009) 521–549, http://dx.doi.org/10.1007/s10270-008-0103-2.
[64] P. Simons, I. Niemelá, T. Soininen, Extending and implementing the stable model semantics, Artif. Intell. 138 (1–2) (2002) 181–234, http://dx.doi.org/10.1016/S0004-3702(02)00187-X.
[65] L. Yngve, W. Xiaoliang, M. Florian, B. Øyvind, S. Anders, R. Adrian, Dpf workbench: a multi-level language workbench for MDE, in: Proceedings of the Estonian Academy of Sciences, 2013, vol. 62, 2013, pp. 3–15. http://dx.doi.org/10.3176/proc.2013.1.02.
[66] J.-P. Tolvanen, S. Kelly, Metaedit+: defining and using integrated domain-specific modeling languages, in: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, OOPSLA '09, ACM, New York, NY, USA, 2009, pp. 819–820. http://dx.doi.org/10.1145/1639950.1640031.
[67] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, P. Volgyesi, The generic modeling environment, in: Proceedings of WISP'2001, WISP'2001, IEEE, 2001.
[68] R.C. Gronback, Eclipse Modeling Project: a Domain-Specific Language (DSL) Toolkit, 1st edition, Addison-Wesley Professional, 2009.
[69] Eclipse Foundation, Sirius, ⟨http://www.eclipse.org/sirius/⟩, 2014.
[70] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S.V. Mierlo, H. Ergin, AToMPM: a web-based modeling environment, in: Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013), vol. 1115, 2013, pp. 21–25.
[71] J. Lara, H. Vangheluwe, AToM3: a tool for multi-formalism and meta-modelling, in: R.-D. Kutsche, H. Weber (Eds.), Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science, vol. 2306, Springer, Berlin, Heidelberg, 2002, pp. 174–188, http://dx.doi.org/10.1007/3-540-45923-5_12.