Modeling Language Extension in the Enterprise Systems Domain

Colin Atkinson, Ralph Gerbig
Chair of Software Engineering
University of Mannheim
Mannheim, Germany
{*atkinson, gerbig*}*@informatik.uni-mannheim.de*

Mathias Fritzsche
SAP AG
Modeling and Taxonomy
Walldorf, Germany
mathias.fritzsche@sap.com

***Abstract*—As the number and diversity of technologies involved in building enterprise systems continues to grow so does the importance of modeling tools that are able to present customized views of enterprise systems to different stakeholders according to their needs and skills. Moreover, since the range of required view types is continuously evolving, it must be possible to extend and enhance the languages and services offered by such tools on an ongoing basis. However, this can be difficult with todays modeling tools because the meta-models that define the languages, views and services they support are usually hardwired and thus not amenable to extension. In practice, therefore, various workarounds have to be used to extend a tool's underlying meta-model. Some of these are built into the implemented modeling standards (e.g. the UML profile, BPMN 2.0 and ArchiMate 2.0 extension mechanisms) while others have to be applied by complementary, external tools (e.g. model weaving). 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 use-cases currently found in practice today.**

***Keywords*-Multi-level Modeling; Model Language Extension; Orthogonal Classification Architecture; Linguistic Classification; Ontological Classification**

I. 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 particular important for visualizing enterprise systems 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 2.0 [5], the Business Process Model and Notation 2.0 (BPMN) [6] and the 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 standards-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 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 provided 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 "low 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 III we then present these strategies in the context of a small case study from the domain of business process performance modeling. Section IV then introduces an alternative modeling approach, known as multi-level modeling, which we believe provides the optimal approach for model extension, while section V analyses this architecture from the point of view of the strategies and requirements outlined in section II. It also proposes enhancements to current multi-level modeling approaches to address some identified weaknesses. Section VI then illustrates how this enhanced, extension-aware form of multi-level modeling would be

1541-7719/13 $26.00 © 2013 IEEE
DOI 10.1109/EDOC.2013.15

		Evolution Form		
		(1) Enhancement	(2) Augmentation	(1) and (2)
Evolution Mechanism	In-built	- introduction of accidental complexity through meta-model element duplication	+ separation of duties by separating problem domains of host language and extension	+ Pluggable extensions - not standardized across tools and modeling languages - no support for domains featuring multiple classificatlion levels
	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 - no support for domains featuring multiple classificatlion levels
	Model Annotation	- introduction of accidental complexity through meta-model element duplication	+ separation of duties by separating problem domains of host language and extension	+ indepenent of tools or modeling languages + Pluggable extensions - no support for domains featuring multiple classificatlion levels

Figure 1. Different forms of extension and the suitability of extension mechanisms. Plus means advantage, whereas minus means disadvantage.

applied in the context of the case study from section III. Finally, section VII concludes with some final remarks and suggestions for future work.

II. Model language extension

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 so that a user can not only create standard software Classes but also remote classes (e.g. Java beans). 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 enhance 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, in-built extension mechanisms b) meta-model customization and c) model annotation. Each have 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, low coupling and high cohesion, and introduce accidental complexity into models. Figure 1 briefly summarizes the pros and cons of these different approaches, as supported in current modeling frameworks, from the point of view of language enhancement and language augmentation scenarios. These are elaborated further in the rest of this section.

A. 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 weaknesses of most current package mechanisms used in meta-modeling tools is that all defined packages 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 they are not supposed to be used for specific tasks and specific views.

B. Language In-built 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 hardwired). Examples of such techniques are the UML profiling mechanism (based on stereotypes), and the BPMN 2.0 or ArchiMate extension mechanisms.

Two problems are immediately obvious with such approaches. The first is when each tool in a heterogeneous enterprise computing environment defines it's 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 mechanims 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 compliance level. 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 [8].

To use a language's built-in extension mechanisms 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 metamodel. 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 — 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 element 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).

C. 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, [9] presents a light weight model annotation approach that allows structured information (see *Annotated Information* in Figure 2) to be attached to other model elements in a *Target Model*. This linkage is technically achieved by refining the *Weaving Model Meta-Model* defined in [10] 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 refining 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) [11], UML Activity diagrams, ArchiMate diagrams or Event-driven Process Chains (EPC) [12].

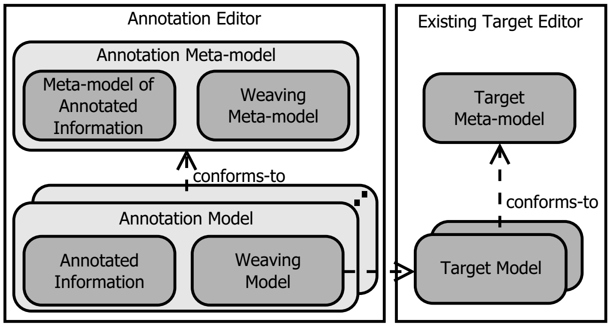
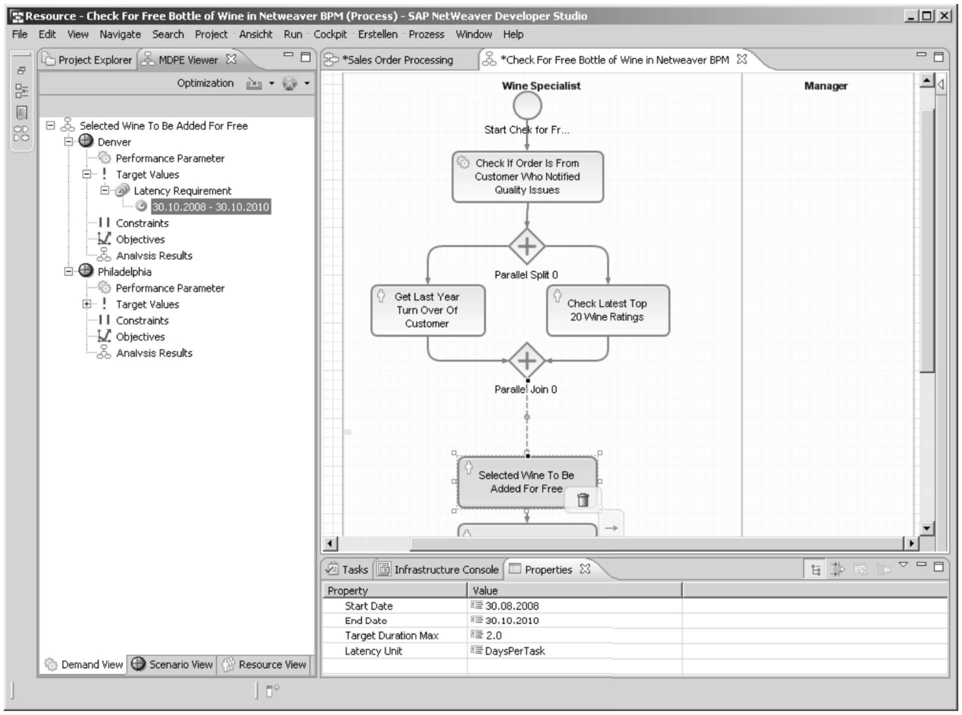


Figure 2. A modeling language extended through annotations.

Given the benefits of this approach, *Annotation Editors* have even been defined to simplify the definition of annotations in [13]. 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 when an annotation should impact the layouting or behavior of an existing target editor, the target editor has to be modified.

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


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	

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

to be introduced which are not needed when employing the meta-model customization approach.

III. 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) [14] 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 is usually stored by business process modeling languages is needed. Moreover, this information comes from a different domain than the domain of 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).

Figure 3 shows a screenshot of the implementation using annotation models developed by Fritzsche [13]. In this screenshot, the BPMN based editor NetWeaver BPM [15] (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 middle MDPE view can show different information depending on what is selected in the model editor. However it has the disadvantage that the host modeling environment used 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. show how the other two extension meachnisms 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 [16], and in the second case they used the UML's built-in extension mechanism based on stereotypes [17]. 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. lead to mutually incompatible extensions of UML and BPMN. In the next sections a technology overcoming these weaknesses will be introduced and demonstrated on the case study of business performance engineering.

IV. 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 (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 compatibility of a universally 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 the tool before using the extended languages.

Such an infrastructure is provided by the so called multi-level modeling approach which is based on the Orthogonal Classification Architecture (OCA) [18]. 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 Figure 4. In this example the linguistic classification levels are stacked vertically while the ontological levels are stacked horizontally within the L_1 linguistic levels.

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 giving rise to so called live meta-modeling [19]. Besides the standard meta-modeling capabilities, multi-level modeling allows domain-specific modeling languages 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 [20] display the level classified by it using this representation.

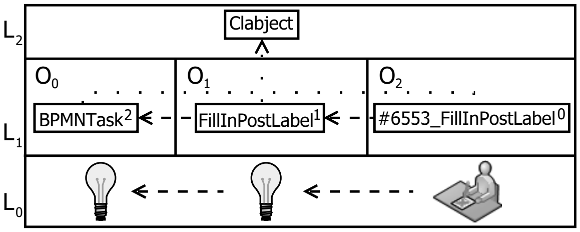


Figure 4. Orthogonal classification architecture instantiated with a business process model.

Figure 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, and instances of model elements with potency 0 cannot exist. In the latter case, instances can have either $*$ potency or a non-negative integer value. Figure 4 shows all model elements at level O_0 with potency 2 and their instances at level O_1 with potency 1. The model elements at O_2 have a potency of 0 which is one lower than the potency of the types at O_1.

The figure also shows that all 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 types (classes) for their instances in the following levels.

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 at 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_1, is used. This supports a general purpose concrete syntax which resembles the UML and ER diagrams. 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 [20].

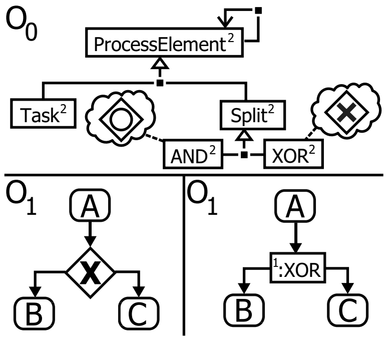


Figure 5. Symbiotic language support.

Figure 5 shows an enterprise computing scenario where this symbitoic 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 [21], built on this architecture, make it possible for a modeller 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 BPMN novice who is not familiar with the difference between the BPMN *Splits* 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.

V. Language Extension in Multi-level Modeling

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

A. Modeling Language Enhancemment

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 Figure 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, the only tool that supports multi-level visual modeling at the current time, provides an emendation service [19] which monitors the model for changes and updates all the levels to make them consistent after a change has occurred.

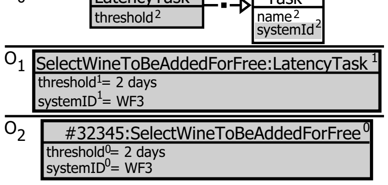


Figure 6. A business process modeling language enhanced by a *LatencyTask* model element and *systemID* attribute.

Figure 6 shows schematically how the multi-level modeling approach inherently supports language enhancement through (meta-)model customization. The right side of the level O_0 represents the core language (e.g. as described in a standard), while the left side 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). So in this example, the general concept *Task* is specialized by *LatencyTask* using inheritance. This introduces the opportunity to model tasks which must finish within a certain time frame. Additionally the process modeling language's *Task* has been enhanced with a *systemId* attribute to allow it to be executed in a certain workflow engine requiring such an attribute. In Figure 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 below (i.e. *#32345*). Note that with the vast majority of tools supporting meta-modeling customization (i.e. so called meta-modeling tools such as MetaEdit+ [22]), 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 *LatencyTask*.

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 is avoided. Provided that a moderately rich designation approach is available, 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 a ubiquitous form but not using ad hoc extension concepts such as stereotypes and tagged value, but in the form of the in-built 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). 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. Concluding multi-level modeling allows modeling languages to be enhanced by introducing less complexity but having the same power as traditional enhancement mechanisms such as UML stereotypes.

Although multi-level modeling provides natural, inherent support for language enhancement, in its pure form, as currently supported by tools such as MelanEE, it does not provide all the features necessary to support modeling language augmentation. As the example above 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 augmentation packages can be included or excluded, as desired, on-the-fly. To support this a dynamic, multi-level-aware packaging mechanism needs to be added.

B. Modeling Language Augmentation

In this section we discuss what such a package mechanism, motivated by the annotation model and UML Profile approaches, should look like and how it can be seamlessly added to multi-level modeling. The purpose of an augmentation package 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 packages 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 a package or hide them. This provides modelers with the option of using the plain host language or the extended language. In contrast to annotation models 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. When deciding to ignore an augmentative package when viewing a model level, 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.

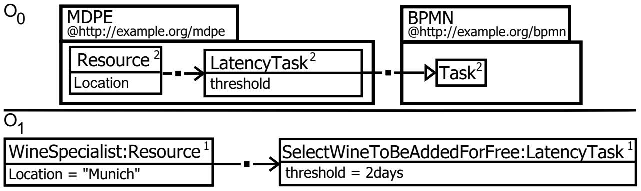


Figure 7. A BPMN package is augmented with *LatencyTask* and *Resource*.

Figure 7 shows how a multi-level modeling environment, enhanced with such dynamically importable augmentation packages, 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 augmentation packages for *BPMN* and *MDPE*. Both packages are remote packages which is indicated by the "@" character followed by a URI underneath the package's name. The *BPMN* package is loaded from *http://example.org/bpmn* and the *MDPE* from *http://example.org/mdpe*. Local packages are indicated by leaving out the location information. The generalization between packages is stored in the local ontological level, which in this case is O_0. It is important that statements about the relationships between packages do not belong to the packages themselves so that they can remain decoupled from one another and can exist independently. Transformations defined on content of a package can be reused after wiring packages together because the model elements on which the transformation is defined remain alive. For example, in Figure 7 a transformation of a business process to a work flow engine format would still work after using the model elements from the augmentation package as these inherit from the types the transformation was defined on.

An example in which a modeling language is extended with different language augmentations is shown in Figure 8. Here, a business process modeling language is extended with information from two different domains at the same time — MDPE information from the *MDPE* package and security information from the *SecurityPackage*. From this information four views can be generated. The most obvious view is the view on *SelectWineToBeAddedForFree* which is an instance of *Task*, *LatencyTask* and *SecurityTask* (Figure 8 (2)). The *SelectWineToBeAddedForFree* instance of *Task*, *LatencyTask* and *SecurityTask* shows information about business process performance and security at the same time. To resolve a possible rendering conflict, the closest abstract supertype is used to render the clabject which in this case is *Task*. The other possible views are the Business Process only view (Figure 8 (3)), the MDPE view (Figure 8 (1)) and the Security view (Figure 8 (4)). They all show different aspects of the overall model, each incorporating information from a different augmentation package. 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.

The next chapter applies this approach to the case study of model-driven business process engineering.

VI. Case Study with Multi-level Modeling

The ontological level O_0 in Figure 8 shows how the example described in section III would be handled using the extension-aware form of multi-level modeling described in the previous section. To augment a language with performance simulation information a user would create a package containing the same content as used in the annotation model described previously. In this example a performance task and resource are modeled. This package can then be loaded into the appropriate ontological level and connected to the original process modeling language through standard language constructs such as specializations or connections. 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 a modeler. The visualization and behavior of model elements of the extended modeling language can be completely overridden using the domain-specific language capabilities supported by multi-level modeling. An augmented language with a completely new look and feel can thus be created with ease by a modeler. In the example, the only change made to the concrete syntax of the DSL is 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. Transformations which are defined on the original language still work with the augmentations as the rules defined on the base types are still valid for the subtypes added through inheritance. Thus, the extended model will run on a workflow engine using the standard version of the model understood by the workflow engine.

To illustrate the multi-view rendering mechanism, a security model has been added to the process modeling language in addition to the performance modeling package in Figure 8. Many more additional augmentation packages can be added the same base language with more augmentations. To easily work with a model containing so much information a modeler can reduce complexity by using the previously described view mechanism. The arrows — (1), (2), (3), (4) — show the different views on the model that are available. 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 latency task. It overrides the concrete default syntax of the business process modeling language's task construct to indicate that a latency task is being shown. This is achieved by displaying a small *L* in the upper right corner of the task. The next view (2) 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. A result of such an analysis could be that a task takes extraordinary long because it generates high business value and is thus especially secured. 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

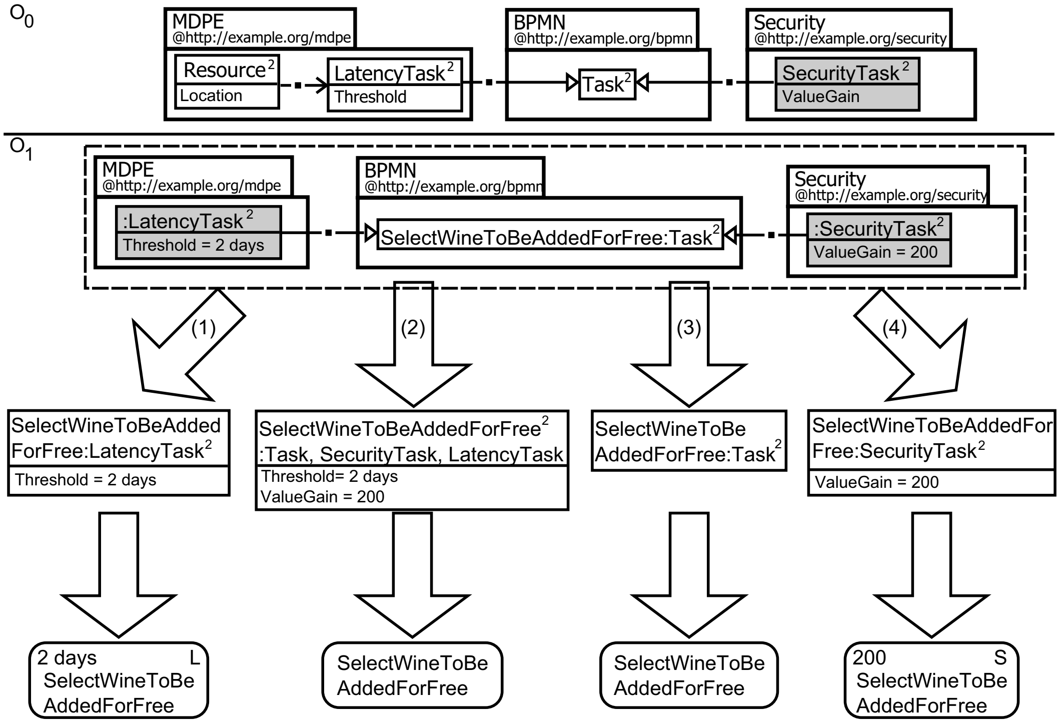

O0
MDPE
@http://example.org/mdpe
Resource²
Location
LatencyTask²
Threshold
BPMN
@http://example.org/bpmn
Task²
Security
@http://example.org/security
SecurityTask²
ValueGain
O1
MDPE
@http://example.org/mdpe
:LatencyTask²
Threshold = 2 days
BPMN
@http://example.org/bpmn
SelectWineToBeAddedForFree:Task²
Security
@http://example.org/security
:SecurityTask²
ValueGain = 200
(1)
(2)
(3)
(4)
SelectWineToBeAddedForFree:LatencyTask²
Threshold = 2 days
SelectWineToBeAddedForFree² :Task, SecurityTask, LatencyTask
Threshold= 2 days
ValueGain = 200
SelectWineToBeAddedForFree:Task²
SelectWineToBeAddedForFree:SecurityTask²
ValueGain = 200
2 days L
SelectWineToBeAddedForFree
SelectWineToBeAddedForFree
SelectWineToBeAddedForFree
200 S
SelectWineToBeAddedForFree

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

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 security task displayed via a custom visualization. The *S* in the upper right of the figure indicates that a security task is shown. The *200* in the upper left of the task symbol indicates that a business value of 200 has been added to the information by processing it through this task.

VII. Conclusion

In this paper we examined the different modeling language extension strategies currently supported in mainstream modeling environments in the domain of enterprise systems, and evaluated their suitability for supporting two key language extension use-cases — 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 deployment 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. Their 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. However, in this paper 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 packaging mechanism capable of supporting the dynamic importing 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.

Based on the insights gained in this paper we are currently upgrading our multi-level modeling tool, MelanEE to support the presented dynamically selectable packaging mechanism, and thereby offer a modeling environment free of any of the weakness 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 II 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 Archimate, TOGAF and other multi-view enterprise visualization standards.

References

[1] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[2] M. Lankhorst, H. Proper, and H. Jonkers, "The architecture of the archimate language," in *Enterprise, Business-Process and Information Systems Modeling*, ser. Lecture Notes in Business Information Processing, T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, and R. Ukor, Eds. Springer Berlin Heidelberg, 2009, pp. 367–380.

[3] The Open Group, "Togaf version 9," http://www.opengroup.org/togaf/, 2009.

[4] J. A. Zachman, "A framework for information systems architecture," *IBM Syst. J.*, vol. 26, no. 3, pp. 276–292, 1987.

[5] 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, and P. Valduriez, "First experiments with a modelweaver," 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] T. O. Group, "ArchiMate 2 Tool Certification Register," http://www.opengroup.org/certifications/archimate/ts-register, 2013.

[9] M. Fritzsche, J. Johannes, U. Aßmann, S. Mitschke, W. Gilani, I. T. A. Spence, T. J. Brown, and P. Kilpatrick, "Systematic usage of embedded modelling languages in automated model transformation chains," in *SLE*, 2008, pp. 134–150.

[10] M. D. D. Fabro, "Metadata management using model weaving and model transformation," in *PhD thesis*, 2007.

[11] 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, and 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.

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

[13] M. Fritzsche, "Performance related decision support for process modelling," in *PhD thesis*, 2010.

[14] M. Fritzsche, M. Picht, W. Gilani, I. T. A. Spence, T. J. Brown, and P. Kilpatrick, "Extending bpm environments of your choice with performance related decision support," in *BPM*, 2009, pp. 97–112.

[15] SAP, "Components & Tools of SAP NetWeaver: SAP NetWeaver Business Process Management," http://www.sap.com/platform/netweaver/components/sapnetweaverbpm/index.epx, download on 31th March 2012.

[16] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "A bpmn extension for the modeling of security requirements in business processes," *IEICE - Trans. Inf. Syst.*, 2007.

[17] A. Rodríguez, E. Fernández-Medina, J. Trujillo, and M. Piattini, "Secure business process model specification through a uml 2.0 activity diagram profile," *Decision Support Systems*, vol. 51, no. 3, 2011.

[18] C. Atkinson, M. Gutheil, and B. Kennel, "A Flexible Infrastructure for Multilevel Language Engineering," *IEEE Transactions on Software Engineering*, 2009.

[19] C. Atkinson, R. Gerbig, and B. Kennel, "On-the-fly refactoring of multi-level models," in *Submitted to ECMFA*, 2012.

[20] 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.

[21] C. Atkinson and R. Gerbig, "Melanie: multi-level modeling and ontology engineering environment," in *Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards*, ser. MW '12. New York, NY, USA: ACM, 2012, pp. 7:1–7:2.

[22] J.-P. Tolvanen and M. Rossi, "Metaedit+: defining and using domain-specific modeling languages and code generators," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ser. OOPSLA '03. New York, NY, USA: ACM, 2003, pp. 92–93.