A Generic Framework for Modifying and Extending Enterprise Modeling Languages

Richard Braun and Werner Esswein
TU Dresden, Chair for Wirtschaftsinformatik, esp. System Development, 01062 Dresden, Germany

Keywords: Meta Modeling, Extensibility, Enterprise Modeling, Language Dialects, Ad Hoc Modification, BPMN.

Abstract: Conceptual modeling languages are of great importance within information systems management. During the last decade, a small set of commonly used enterprise modeling languages established and gained broad acceptance in both academia and practice (e.g., BPMN). Due to their dissemination, these languages often need to be extended or adapted for domain-specific or technical requirements. Since most modeling languages provide rather poor extension mechanisms, it is necessary to modify a language meta model directly. However, there is lack of integrated methodical support for these modifications. Within this position paper, we therefore proclaim a generic framework for modifying enterprise modeling languages on the meta model level. The framework is divided into the main parts of a modeling language (abstract syntax, concrete syntax, semantics) and respective operations (add, remove, specify and redefine).

1 INTRODUCTION AND MOTIVATION

Enterprise Modeling Languages (EML) are of primary importance for conceptual and technical complexity management within enterprises as they facilitate an integrated management of views, aspects and levels of abstraction (Frank, 1999; Frank, 2002; Braun and Winter, 2005). Examples for such EMLs are BPMN (OMG, 2011a), ARIS (Scheer and Nüttgens, 2000) or MEMO (Frank, 2002). Additionally, the more engineering-driven general-purpose modeling language UML is frequently adapted in the field of enterprise modeling (Silingas and Butleris, 2009). The stated languages can largely be seen as de facto standards due to their prevalence and acceptance both in academia and professional practice [1]. This level of dissemination implies several benefits such as a common understanding of syntactical rules and semantical interpretations. It can be also assumed, that these languages are subject to quality management and evolvement over time. However, there are also some challenges: As it is well known from other fields (e.g., ERP management), the emergence of commonly used artifacts leads often to an increasing demand for individual customization in order to satisfy specific requirements coming from the peculiarities of a project, business or industry (Braun and Esswein, 2014; Pardillo, 2010). Therefore, it seems to be extremely relevant to consider the issue of extending or modifying EMLs for domain-specific adaptations, model transformation support, interoperability and language evolution (Braun, 2015b). Further, language extensions and modifications are contributions to the management of language pluralism. Instead of springing up "yet another" domain-specific modeling language (DSML), having limited practical relevance, we proclaim the usage of a rather limited set of common EMLs on the one hand, while focussing their enhancement, modification and extensibility on the other hand (Atkinson et al., 2013; Braun, 2015b) [2].

Extensions or modifications of EMLs are usually employed on the meta model level of a language, which is referred as M2 level (OMG, 2014). Modifications lead to particular meta model variants of a considered modeling language (Esswein and Weller, 2007). It is often necessary to address the meta model level, as the majority of EMLs either does not provide any extension mechanism or existing mechanisms reveal some architectural deficits (Braun, 2015b; Braun, 2015a). Even built extensions of languages with

[1] Indeed, BPMN and UML are specified as official ISO standards (OMG, 2011b; OMG, 2011a).

[2] Nevertheless, we do not want to deny the powerfulness and accuracy of DSMLs at all. DSMLs are valuable for very specific problems or projects.

Braun R. and Esswein W..
A Generic Framework for Modifying and Extending Enterprise Modeling Languages.
DOI: 10.5220/0005351702770286
In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS-2015), pages 277-286
ISBN: 978-989-758-097-0
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

rather well-defined extension interfaces (e.g., BPMN) are mainly not designed in accordance to them (Braun and Esswein, 2014). While this issue might be unproblematic within single projects, it hampers interoperability, change tracking and general comprehension of the entire modification process. Further, it is necessary to regulate modeling language changes in order to avoid blowing up or destruction of a language (Häggmark and Ågerfalk, 2006).

Generally, this deficiency could be tackled by designing consistent and integrated extension mechanisms within EMLs. With regard to the common four level meta model architecture (OMG, 2014; Strahringer, 1996), this proposal actually causes a problem shift to the meta meta model level (M3 level), as the M3 level should define appropriate concepts and constraints for extending a modeling language that is defined on the meta model level (level M2). In particular, this aim exemplarily causes a revision of large parts of the Meta Object Facility (MOF), as MOF is the most common meta modeling language (OMG, 2014). However, this is problematic facing the dissemination of the current MOF version and the rather crude architecture of some parts of the MOF itself (Braun, 2015b).

Hence, it seems to be much more feasible and viable to tackle the issue on level M2 by providing a reference framework that supports the systematic and comprehensible design of EML modifications. More efficient design of modifications can also lead to lower design costs at all. Therefore, this research-in-progress article aims to outline a generic framework for modifying meta models of EMLs. Consequently, the remainder of this article is as follows. Section 2 provides some fundamentals from the field of conceptual modeling and introduces the E3 model of GREIFFENBERG (2004), which is adapted for the structure of our framework. Section 3 presents the entire framework and describes each part of the framework in detail. The article ends with a short summary and an outlook to further research.

2 FUNDAMENTALS

2.1 Conceptual Modeling

A *conceptual model* is defined as the result of a construction process, "done by a modeler, who examines the elements of a system for a specific purpose" (Schuette and Rotthowe, 1998). In contrast to design models, which typically represent software systems or parts of it, conceptual models represent and outline real world phenomena (Wand and Weber, 2002). Thereby, graphical or diagrammatic conceptual models have been established as an appropriate medium to foster communication between business stakeholders within the information system discipline (Frank, 1999). Thus, EMLs are usually *semi-formal modeling languages*. These languages combine aspects of formal languages and natural languages. The syntax of semi-formal modeling languages is defined formally within a meta model, but the semantics come from specific domain terminology that is typically stated in a natural language (Pfeiffer and Gehlert, 2005, p. 111). According to that, a *conceptual modeling language* is the result of the application of a specific terminology to a meta model based conceptual modeling grammar (Wand and Weber, 2002, p. 364). Modeling languages should always be embedded into a *modeling method*, which consists of the modeling language itself (represented by a meta model) and a procedure delineating the process of model building and model usage (Greiffenberg, 2004). Thereby, the *meta model* is defined as a model representing a modeling language (Strahringer, 1996). As stated at the beginning of this article, meta models are typically located at the M2 level of the common four layer architecture of MOF (OMG, 2014).

As stated in Section 1, this article considers modifications of a language meta models due to domain-specific alterations or other business-driven purposes. Alterations of a language meta model can be caused by the integration of new diagrams or perspectives (e.g., (Stroppi et al., 2015)), the integration of new domain-specific concepts (e.g. (Braun and Esswein, 2014)) or even user-specific reductions of a language (based on (zur Muehlen and Recker, 2008)). Generally, it can be stated that requirements on conceptual models are frequently changing and it is often necessary to work on different language versions in parallel (Ralyté et al., 2004). Therefore, we define a *meta model modification* as any kind of meta model alteration, which does not apply possibly existing extension mechanisms (Esswein and Weller, 2007; Braun, 2015b). The application of an altered modeling language, having a notable set of new or changed elements, that is used by a group of model users leads to a *modeling language dialect* [3]. In addition to the frequently used term *extension*, we want to emphasize, that even meta model reductions and minor meta model alterations are understood as meta model modifications, since this article is inspired by the works from the field of meta model configuration manage-

[3]Besides, it is important to emphasize the need for a more precise definition of language dialects. For instance, in terms of both type and quantity of concepts which lead to an distinction from the original language.

ment (Esswein and Weller, 2007).

2.2 E3 Model as Frame of Reference

The smallest unit of modification can be referred as configuration item (Esswein and Weller, 2007). Such a configuration item can stand for all elements that are specified within a meta model. In preparation of the aimed framework, it is required to typecast these elements in a generic way and under consideration of *all* language parts (abstract syntax, concrete syntax and semantics). Examining the common MOF meta modeling language reveals some shortcomings regarding to the integrated covering of these aspects, as MOF solely focusses abstract syntax and suffers from some architectural problems (e.g., mixing up language and tool definitions or self-definition by the UML infrastructure). Hence, an alternative meta modeling languages was selected: The E3 language of GREIFFENBERG (2004). In contrast to similar approaches such as MOF (OMG, 2014) or Gmodel (Bettin and Clark, 2010), E3 benefits from its integrated and cohere architecture, which is presented in Figure 1.

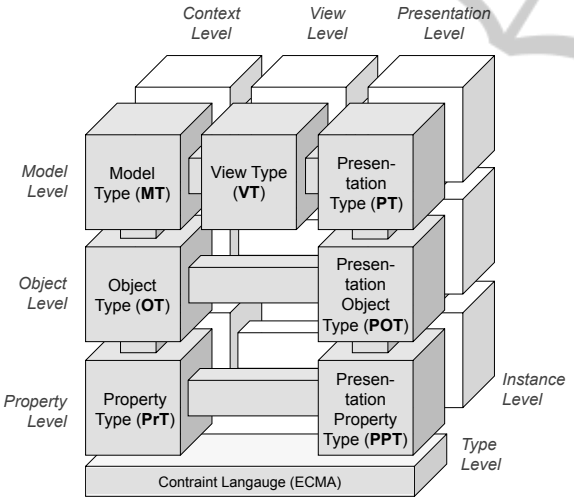


Figure 1: Architecture of the E3 meta modeling language (Greiffenberg, 2004) that is adapted for the structure the modification framework.

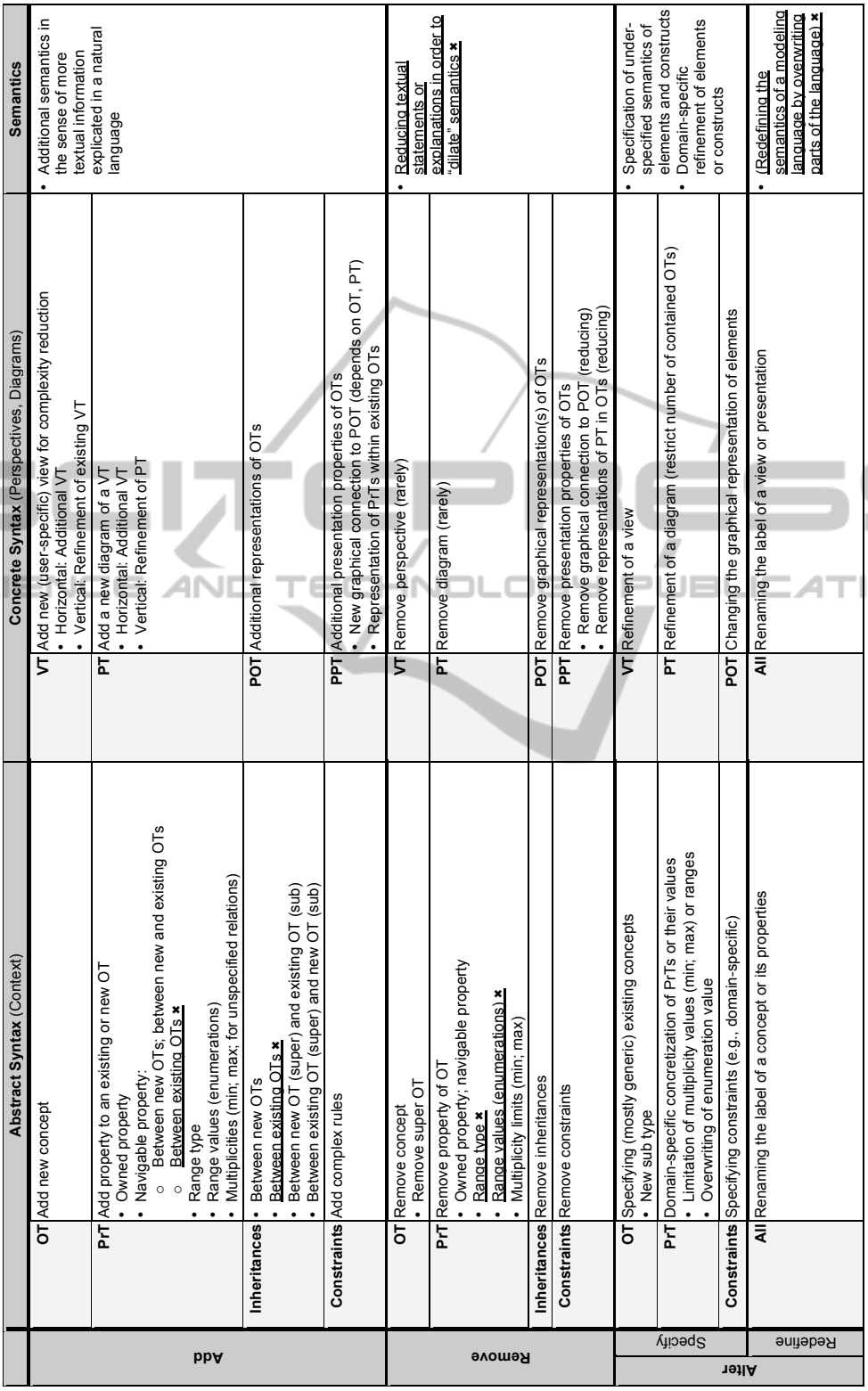
The E3 architecture is divided into a type layer (meta meta model) and an instance level (meta model), whereby each instantiated element relates to exactly one type element. Horizontal levels of E3 provide concepts for the definition of hierarchical perspectives and diagrams to the entire model context for reasons of user-oriented complexity reduction (Greiffenberg, 2004, p. 116). Thus, it is divided into the *Context Level*, the *View Level* and the *Presentation Level*, whereby a particular level is bound to the next level on the left side. It means, that a *Model Type (MT)* (e.g., UML) owns several *View Types (VT)* (e.g., static and dynamic views), which in turn refer to particular *Presentation Types (PT)*. For instance, class diagrams are presentations of the static view and sequence diagrams are presentations of the dynamic view within UML. Hence, E3 supports separation of concerns between context and presentation while ensuring their integration. Vertically, E3 is divided into the *Model Level*, the *Object Level* and the *Property Level* (Greiffenberg, 2004, p. 113). The level of abstraction descends from the top to the bottom. Within the model level, the most abstract containers are specified: *Model Types*, *View Types* and *Presentations Types*. Below, specific elements in form of *Object Types (OT)* and their corresponding *Presentation Object Types (POT)* can be defined in order to express all concepts and properties of a language.

Object Types represent intended concepts of a language (e.g., UML classes). *Presentation Object Types (POT)* represent (perhaps multiple) graphical representations of theses elements (e.g., expanded and collapsed UML classes). It is important to note, that both concepts and relations are understood as *Object Types* in E3 (Greiffenberg, 2004, p. 101). The lowest horizontal layer describes properties of *Object Types* (*Property Types*) and *Presentation Types* (*Presentation Property Types*). *Property Types (PrT)* define particular ranges that can be constituted as simple ranges (e.g., String, Integer, Boolean) or complex ranges (references to other *Object Types*). Each *Property Type (PT)* owns a specific cardinality (min, max) and a structure type for managing its instances on a technical level (Greiffenberg, 2004, p. 102). *Presentation Property Types (PPT)* are used for the definition of specific attributes of graphical elements (e.g., connector points between elements).

In addition to the introduced structural elements, E3 provides an integrated ECMAscript based constraint language for the specification of complex rules and constraints both within abstract syntax (e.g., triggers for object renaming or consistency checks) or concrete syntax (e.g., context-aware graphical adaptions).

3 FRAMEWORK ARCHITECTURE

This section provides the proposed framework for conducting meta model modifications based on the E3 architecture. The framework is vertically divided into the three components of modeling languages. *Abstract Syntax* covers all elements, rules and constraints of a language (Wand and Weber, 2002) and



		Abstract Syntax (Context)	Concrete Syntax (Perspectives, Diagrams)		Semantics
Add	OT	Add new concept	VT Add new (user-specific) view for complexity reduction • Horizontal: Additional VT • Vertical: Refinement of existing VT		• Additional semantics in the sense of more textual information explicated in a natural language
	PrT	Add property to an existing or new OT • Owned property • Navigable property: ○ Between new OTs; between new and existing OTs ○ Between existing OTs **×** • Range type • Range values (enumerations) • Multiplicities (min; max; for unspecified relations)	PT Add a new diagram of a VT • Horizontal: Additional VT • Vertical: Refinement of PT		
	Inheritances	• Between new OTs • Between existing OTs **×** • Between new OT (super) and existing OT (sub) • Between existing OT (super) and new OT (sub)	POT Additional representations of OTs		
	Constraints	Add complex rules	PPT Additional presentation properties of OTs • New graphical connection to POT (depends on OT, PT) • Representation of PrTs within existing OTs		
Remove	OT	Remove concept • Remove super OT	VT Remove perspective (rarely)		• Reducing textual statements or explanations in order to "dilate" semantics **×**
	PrT	Remove property of OT • Owned property; navigable property • Range type **×** • Range values (enumerations) **×** • Multiplicity limits (min; max)	PT Remove diagram (rarely)		
	Inheritances	Remove inheritances	POT Remove graphical representation(s) of OTs		
	Constraints	Remove constraints	PPT Remove presentation properties of OTs • Remove graphical connection to POT (reducing) • Remove representations of PT in OTs (reducing)		
Alter / Specify	OT	Specifying (mostly generic) existing concepts • New sub type	VT Refinement of a view		• Specification of under-specified semantics of elements and constructs • Domain-specific refinement of elements or constructs
	PrT	Domain-specific concretization of PrTs or their values • Limitation of multiplicity values (min; max) or ranges • Overwriting of enumeration value	PT Refinement of a diagram (restrict number of contained OTs)		
	Constraints	Specifying constraints (e.g., domain-specific)	POT Changing the graphical representation of elements		
Alter / Redefine	All	Renaming the label of a concept or its properties	All Renaming the label of a view or presentation		• (Redefining the semantics of a modeling language by overwriting parts of the language) **×**

Figure 2: Proposed modification framework (thick underlinings and cross symbols emphasize crucial modifications; cf. Section 3.2).

refers to the previously introduced *Context Level* of E3. *Concrete Syntax* stands for the graphical representation of language concepts and their relationships and refers both to the *View Level* and the *Presentation Level* of E3. The*Semantics* column covers the meaning of modeling language constructs in application domains (Pfeiffer and Gehlert, 2005). Semantics are usually expressed separately by natural language statements, domain dictionaries or glossaries.

Vertically, the framework is divided into those operations, which can be applied to an element within the meta model: *Add*, *Remove* and *Alter*. Model element alterations are further divided into *Specifying* and *Renaming*. Alteration by specification refers to the concretization of generic or under-specified model elements (Braun, 2015b). Alteration by redefining refers either to simple element renaming or to the redefinition of their semantics. Figure 2 represents the modification framework by assigning particular E3 elements to segments of the framework matrix.

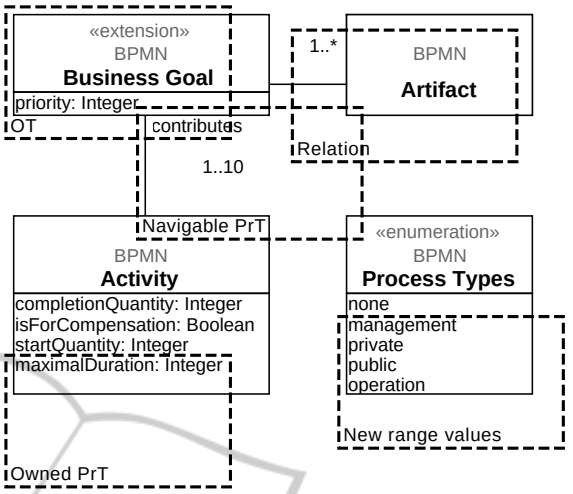


Figure 3: Basal example depicting additional owned and navigable Property Types (PT), a new Object Type (OT) as well as extended range values within an enumeration.

3.1 Framework Segments

Below, each segment of the framework is presented in detail by explaining its implications on meta model elements and possible reasons for adaptation of specific alteration techniques. Relevant contributions of other authors are explicitly stated in order to enable a seamless integration with existing techniques. We also consider impacts of meta model changes on already instantiated models on level M1 in order to keep them consistent. Therefore, recommendations for model management are given and points for manual processing are explicated. Additionally, several examples are given by using the popular business process modeling language BPMN (OMG, 2011a).

3.1.1 Add Abstract Syntax

Adding new features to a modeling language constitutes as adding concepts and properties to a meta model in order to enhance its expressiveness (Atkinson et al., 2013). Basically, adding a new OT to a meta model means contextual enrichment of a language with a delimitable concept that is not represented so far. Typically, this is used for dilating the scope of a language horizontally (while *specifying* merely refers to a vertical kind of detailing). In contrast to the implementation of additional OTs, adding new properties is quite diverse as there are plenty ways for implementation. First, owned PrTs can be added to OTs in order to realize a more detailed specification of existing OTs (e.g., by attaching domain-specific information). Also, simple ranges of existing properties can be extended by new range types or by declaring extra range values (e.g., further enumeration values). Second, navigable properties (references) can be added between newly created or original OTs.

However, it has to be emphasized that relations between original OTs are crucial as they overwrite the intended core attributes of the language itself. Third, multiplicities on unspecified associations can be added in order to introduce stricter limitations between instances of these elements.

Further, it is imaginable to introduce new inheritances between OTs in order to reduce model complexity and improve the level of hierarchical reuse. Again, this is unproblematic between newly added OTs. However, it becomes more complicated when considering possible inheritances between original OTs or introducing a new super OT to an existing (sub) OT. The first case refers to the implementation of hierarchical structures within parts of the original language, while the latter covers the issue of OT specialization. Finally, it is also possible to add new complex rules within the program code of the used constraint language. This can be done due to the implementation of rigid, perhaps business-specific rules.
Consequences. Generally, adding meta model elements is mostly unproblematic, since a previously instantiated model still corresponds to a subset of the extended meta model. Thus, appending new OTs does not require any follow-up action. Adding PTs requires more attention if they are mandatory. In this case, property values have to be assigned to model instances in order to ensure validity. Additional property range types or range values do not require alterations as they only extend the scope of valid ranges. In contrast, adding new multiplicities repre-

sents stronger constraints of models. Lowering minima values and raising maxima values are unproblematic, since the former multiplicity limits are always included. In contrast, increasing minima values and reducing maxima values require modifications. Increasing minima values causes the identification of affected model elements. If necessary, default elements have to be added in order to match the rule. In the case of reducing maxima, surplus relations to other elements have to be reduced manually in order to match the new max value (Esswein and Weller, 2007).

Similar to that, introducing new inheritances usually lead to model alterations. Hierarchical structures between added OTs or existing super OTs and newly added sub OTs are unproblematic and do not require follow-up actions. Inheritances, which create any situation where an original OT received a new super OT, require revisions by assigning all properties of the super OT to the respective sub OT. Afterwards, each added property has to be dealt like a generally added property. Accordingly, it is perhaps necessary to define and add default property values to model instances.

3.1.2 Remove Abstract Syntax

Removing features of a meta model is a fairly hard intervention as it erases original meta model constructs and reduces the expressiveness of the language. While meta model extensions are considered in numerous publications, meta model reduction has not gained comparable exposure, although it is conceived as very important (Fondement et al., 2013, p. 139). Motivated by the complexity of the UML meta model, BAE ET AL. (2008) introduce the concept of meta model slicing that intends to cut out selected classes, properties and their transitively dependent elements in order to keep only user-relevant meta model parts (Bae et al., 2008). Similar to that, SEN ET AL. (2009) proclaim meta model pruning in the sense of removing all unnecessary classes and properties by applying a graph-based removal algorithm (Sen et al., 2009, p. 33).

FONDEMENT ET AL. (2013) introduce the meta model unmerging approach as contradiction of merging; the union-like copy technique from MOF (OMG, 2014). Therefore, eleven unmerging use cases (UC) are proclaimed (Fondement et al., 2013, p. 143). As the unmerging approach provides an integrated framework for nearly all relevant reduction cases, we partly adapt their patterns to our framework: Removal of OTs refers to UC3 (classes); removal of super OTs refers to UC7 (hierarchies); removal of owned properties refers to UC5; removal of navigable properties refers to UC6; removal of multiplicity limits refers to UC11 and removing constraints refers to UC10 (Fondement et al., 2013). Additionally, we propose removing range types and range values. Thus, it might be possible to release some constraints or remove unused range values.

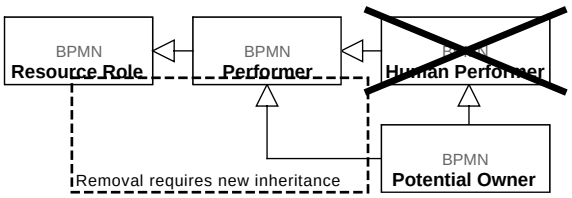


Figure 4: Removal of an object type requires the integration of a new inheritance (OMG, 2011a, p. 154).

Consequences. If an OT is removed from the meta model, its instances need to be deleted, too. Besides, references to the respective OT need to be removed (both on type and element level). Those references can be occurred as properties or constraints. If an OT was a super OT within a hierarchy, it is necessary, to re-organize the meta model hierarchy in order to ensure hierarchical relations of sub OTs to all their super OTs in order to bridge the created gap (Fondement et al., 2013, p. 143). On model level, instances of sub OTs of the removed OTs need to be updated recursively in terms of removing inherited properties. If any kind of a PrT is removed from the meta model, its corresponding properties on model level are also removed. Removing a specific range type is indeed allowed, but leads to unspecified properties. We thus recommend the specification of a default range type. If range values (enumerations) are removed, it is also suggest to define a default enumeration value in order to avoid null values. Removing multiplicity values (min, max) or constraints requires no follow-up actions, since the model is than less constrained at all and still valid.

3.1.3 Specify Abstract Syntax

Specifying the abstract syntax encompasses specific configuration of intentionally generic elements or those elements, which are perceived as overly generic in the modeling context (Braun and Esswein, 2014, p. 47). Such specialization is mostly implemented by introducing new sub OTs owning domain-specific PrTs. On the level of PrTs, it might be appropriate to specify multiplicity values or to limit ranges (e.g., Integer instead of Float). Further, specific range values can be overwritten if the previous notation is too broad [4]. In contrast to modification by addition, spec-

[4] Actually, each alteration is a combination of adding or even removing from a logical point of view. For reasons of a purposive representation, we refrain from that separation.

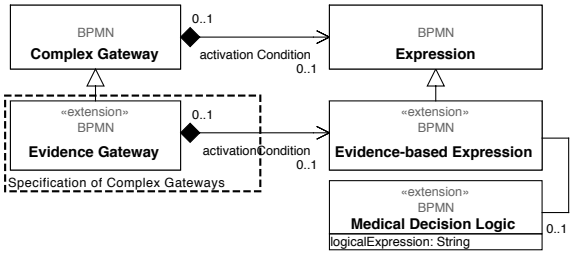


Figure 5: Specifying elements by inheritances (adapted from the BPMN4CP extension (Braun et al., 2014, p. 15)).

ifying has a much stronger relation to the scope of the meta model and its limitations.

Consequences. Specification of OTs by introducing new sub OTs does not necessarily require follow-up actions, since the model is still valid. However, a redefinition of those elements that are instantiated from the specified generic OT might be rational. Redefinitions of multiplicity values have to be treated as described in Section 3.1.1. If enumeration values are overwritten, a matching list is proposed in order to comprehend respective updates.

3.1.4 Redefine Abstract Syntax

Redefining abstract syntax covers the case of renaming an element (Esswein and Weller, 2007). Such redefinitions are mostly conducted for a better model understanding in a specific user environment or avoiding natural language based misunderstandings and ambiguities. Therefore, renaming is tightly coupled to semantics as it refers to real-world concepts. There are no direct implications on model level.

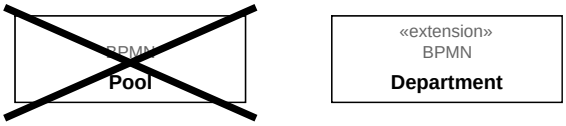


Figure 6: Rename the *Pool* concept in order to express organizational units on type level.

3.1.5 Add Concrete Syntax

Modification of the concrete syntax is rarely considered in literature so far. With regard to the above mentioned E3 meta framework, the concrete syntax can be described by *View Types* (VTs, filtering perspectives), *Presentation Types* (PTs, diagrams), *Presentation Object Types* (POTs, graphical constructs) and corresponding *Presentation Property Types* (PPTs).

Adding a new VT is either used for adding better complexity reduction mechanisms within a large set of OTs or for better user-specific segmentation of the context (similar to separation of concern (Atkinson et al., 2013, p. 47)). Such a filtering can be implemented horizontally or vertically. The *horizontal type* covers the definition of further VTs on the set of all OTs within a model type. The *vertical type* refers to an existing VT and refines them. Additional PTs (diagrams) are implemented in a similar way. *Horizontal* PT addition refers to new PTs within a VT; in parallel to other diagrams. *Vertical* PT addition addresses the refinement of existing diagrams, what is tightly coupled to VT additions. For instance, STROPPI ET AL. (2015) provide additional resource perspectives and diagrams within the BPMN.

Adding new POTs is a quite simple operation that is often provided by modeling tools or even stated in meta model specifications (e.g., (OMG, 2011a)). Thus, it is possible to add alternative graphical representations for OTs in order to support model understanding (e.g., (Stark and Esswein, 2012)). Adding PPTs to a meta model is caused by two purposes. First, is might be necessary to represent PrTs of existing OTs graphically (e.g., property-dependent icons within the graphic). Second, for the graphical representation of relations between original and added POTs.

Consequences. Adding new VTs, PTs, POTs or PPTs does not require dedicated follow-up actions as it only provides more alternatives for model representation.

3.1.6 Remove Concrete Syntax

Removing VTs or PTs is a rather uncommon operation as it presumes a strong segmentation within the meta model, which can be found in the context of enterprise architecture modeling language (Lankhorst et al., 2009). In contrast, removing POTs or particular PPTs is more relevant for several reasons (zur Muehlen and Recker, 2008). First, it is an appropriate means for improving model understandability by reducing the occasionally vast set of graphical elements. Second, it might be promising to reduce some relations between model elements in order to enhance model readability. Removing a POT requires the deletion of PPTs, which originally has implemented graphical references to these POTs. Thus, all references to these POTs have to be identified and removed.

Consequences. Similar to abstract syntax, removing concrete syntax requires various follow-up actions on model level. Removing a VT is mostly unproblematic as it just works as an abstraction filter that is than missing. We propose to assign all PTs of the VT to the next existing VT in the E3 hierarchy. If the VT is located on the topmost layer, containing PTs are directly assigned to the respective MT. The same

procedure should be applied to PTs and containing POTs. Removing POTs is manifested by both erasing all graphics of that POT and erasing all affected PPTs (e.g., removing all ingoing and outgoing edges). If PPTs are removed, all affected graphics have to be updated and reloaded in the new presentation style. Finally, it has to be stated, that all these graphical alterations cause a revision of model positions due to particular re-arrangements (Becker et al., 2007; Esswein and Weller, 2007).

3.1.7 Specify Concrete Syntax

Specifying VTs addresses the case of limiting the number of containing PTs in order to focus on specific diagrams without removing particular PTs. For example, this could be useful in order the emphasize, that the structural view on UML is limited to class and package diagrams. Specifying PTs stands for the limitation of containing POTs in order to focus particular elements. Next to these more uncommon operations, specification of POTs is often used in order to adapt given graphics to specific user environments. A simple example for that is the usage of emphasizing colors or icons that are typical for a specific application, domain or context. In short, typical monochrome rectangles and circles are adapted visually in order to bridge the cognitive gap between a model and its readers.

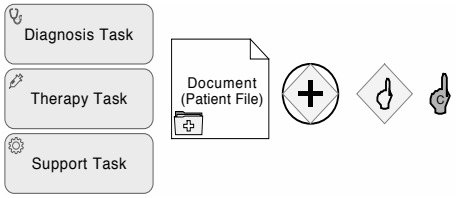


Figure 7: Graphical specifications of *Activities*, *Artifacts* and *Gateways* within BPMN (Braun et al., 2014).

Consequences. Generally, specifications of the concrete syntax can be automatically adapted to its instances (e.g., by updating graphics and diagrams). However, it is often necessary to reposition both models and diagrams due to changing element sizes and dimensions.

3.1.8 Redefine Concrete Syntax

Redefining the concrete syntax is understood as renaming a view or presentation within a meta model in order to increase user consciousness of their meaning. There are no direct implications on model level.

3.1.9 Add Semantics

Additional semantics is understood as providing more contextual information (and thus mostly restrictions) regarding the meaning of particular concepts and constructs by explicating further information in a natural language. Hence, the relation of particular concepts to real-world entities or concepts is sharpened and specified. We therefore divide into *explicit* and *implicit* addition of semantics. The first one addresses the stated extension of textual element descriptions. The second one refers to additional semantics coming from syntactical changes.
Consequences. As the semantics of a conceptual modeling language cannot be formalized, it is not possible to automate any semantical modification on model level. Rather, it has to be proven, whether a model is still consistent in the light of the changed semantics. If not, parts of the model have to be rearranged (Esswein and Weller, 2007).

3.1.10 Remove Semantics

Removing semantics is understood as increasing the degree of freedom regarding to a particular meta model element. This could be seen as simplification of a language in order to provide more interpretations and applications. On the one side, such reduction could be helpful for widening the scope of a language or its concepts. On the other side, it provokes the risk of language softening and defacement (see Section 3.2).
Consequences. There is no need for further follow-up operations as the models are implicitly valid.

3.1.11 Specify Semantics

Specifying semantics covers situations where intentionally under-specified semantics of meta model elements are refined and described in detail by providing more specific contextual information. Hence, the way of model interoperation is usually restricted. This is a simple way of adapting a modeling language for a specific domain without changing the syntax at all. For instance, the *Lane* concept of BPMN is intentionally under-specified and needs to be adapted for a particular model context (OMG, 2011a, p. 308).
Consequences. Each model element, which is instantiated from a meta model element with specified semantics, needs to be checked whether it is still in the tighter scope of meaning or not. If not, it is necessary to update either the title of the element or the model arrangement.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

3.1.12 Redefine Semantics

The redefinition of parts of a modeling language should not be acceptable as it would lead to an immediate defacement of the semantical core of a modeling language. As a modeling language dialect should always enable the identification of the original modeling language, we do not recommend semantical redefinitions. It has to be emphasized, that semantical redefinitions can also occur by enormous adding and removing operations (see above). Thus, it is necessary, to provide strong reasons and evidences for each of these operations in order to avoid excessive alienation from the original language.

3.2 Modeling Language Defacement

As stated at several passages, iterative or even excessive modifications of a language meta model could lead to language defacement or language alienation. It stands for a remarkable estrangement of the original intention of a language and is caused by the possibility of both removing elements and changing its semantics. We therefore distinguish three categories of alienation:

Concept Estrangement. Basically, each alteration of a language element leads to some kind of estrangement. However, we argue that added relations and inheritances between existing OTs are especially cruical as they contradict the original structure and meaning of a modeling language. We therefore argue, that additional inheritances and references between original OTs should be avoided.

Language Reduction. As stated, meta models can be pruned or sliced by respective techniques. However, especially a strong reduction of abstract syntax decreases the intended expressiveness and consistency of the entire language. In order to avoid destruction of the language core (in the sense of the most important and relevant OTs and its relations), it seems to be promising the assess the indispensability of an OT for a language and forbid its removal. Alternatively, is might be appropriate to relax strict constraints within a meta model (e.g., changing mandatory to optional relations) and avoid element removing in general.

Semantics Redefinition. As already stated, explicit semantical changes or redefinitions are crucial if they affect large parts of a language and lead to a decoupling of syntactical elements and their interpretation. We therefore recommend, that only semantical specifications and additional semantics should be applied. Removing or overwriting is not recommended.

4 CONCLUSION

This research-in-progress article propose an integrated framework for meta model modifications of enterprise modeling languages in order to support systematic ad hoc changes. We therefore adapted the E3 meta framework and provide add, remove and change operations for each segment. As the framework is generically applicable to any meta model, it can be used for conducting meta model alterations of modeling languages from different meta model languages. We explicitly focused on the meta model level as most of the adaptions are conducted on that level due to architectural reasons within the most languages. This article also outlines limitations and risks of excessive meta model interventions. Of course, there are several aspects for further research:

In terms of better methodical support, an algorithm for (partly automated) adaption of models after meta model modifications should be designed. As there are several approaches for single purposes (e.g., meta model pruning), it seems to be necessary to integrate them to an entire method according to our proposed framework. It is than possible to extend each meta model based language systematically. Technically, this method should be designed generically but needs to be validated with MOF based languages as MOF is the de-facto standard for meta meta modeling. Further, it is necessary to provide suitable modification patterns for specific modification purposes. Finally, research on modeling language dialects needs to be intensified in general. For instance, it is conceivable to highlight those elements of a meta model that should not be changed in order to keep a stable language core a avoid language defacement.

ACKNOWLEDGEMENT

The presented research results are partial outcome of the research project ”SFB Transregio 96” that is funded by the German Research Foundation (DFG). The authors is grateful for the funding and its related research opportunities.

REFERENCES

Atkinson, C., Gerbig, R., and Fritzsche, M. (2013). Modeling language extension in the enterprise systems domain. In *17th IEEE International Enterprise Distributed Object Computing Conference*, pages 49–58.

Bae, J. H., Lee, K., and Chae, H. S. (2008). Modularization of the uml metamodel using model slicing. In *Information Technology: New Generations, 2008. ITNG*

2008. Fifth International Conference on, pages 1253–1254. IEEE.

Becker, J., Delfmann, P., and Knackstedt, R. (2007). Adaptive reference modeling: integrating configurative and generic adaptation techniques for information models. In *Reference Modeling*, pages 27–58. Springer.

Bettin, J. and Clark, T. (2010). Advanced modelling made simple with the gmodel metalanguage. In *Proceedings of the First International Workshop on Model-Driven Interoperability*, pages 79–88. ACM.

Braun, C. and Winter, R. (2005). A comprehensive enterprise architecture metamodel and its implementation using a metamodeling platform. *Proceedings of the Workshop Enterprise Modelling and Information Systems Architectures*, pages 24–25.

Braun, R. (2015a). Behind the scenes of the bpmn extension mechanism - principles, problems and options for improvement. In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development*.

Braun, R. (2015b). Towards the state of the art of extending enterprise modeling languages. In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development*.

Braun, R. and Esswein, W. (2014). Classification of domain-specific bpmn extensions. *Lecture Notes of Business Information Processing*, 147:42–57.

Braun, R., Schlieter, H., Burwitz, M., and Esswein, W. (2014). Bpmn4cp: Design and implementation of a bpmn extension for clinical pathways. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 9–16. IEEE.

Esswein, W. and Weller, J. (2007). Method modifications in a configuration management environment. *Proceedings of the Fifteenth European Conference on Information Systems*, pages 2002–2013.

Fondement, F., Muller, P.-A., Thiry, L., Wittmann, B., and Forestier, G. (2013). Big metamodels are evil. In *Model-Driven Engineering Languages and Systems*, volume 8107 of *Lecture Notes in Computer Science*, pages 138–153. Springer Berlin Heidelberg.

Frank, U. (1999). Conceptual modelling as the core of the information systems discipline-perspectives and epistemological challenges. In *Proceedings of the Fifth Americas Conference on Information Systems (AMCIS 1999)*, pages 695–697.

Frank, U. (2002). Multi-perspective enterprise modeling (memo) conceptual framework and modeling languages. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 1258–1267.

Greiffenberg, S. (2004). *Methodenentwicklung in Wirtschaft und Verwaltung*. Kovač.

Häggmark, M. and Ågerfalk, P. J. (2006). Why software engineers do not keep to the principle of separating business logic from display: A method rationale analysis. In *Advanced Information Systems Engineering*, pages 399–413. Springer.

Lankhorst, M. M., Proper, H. A., and Jonkers, H. (2009). The architecture of the archimate language. In *Enterprise, Business-Process and Information Systems Modeling*, pages 367–380. Springer.

OMG (2011a). *Business Process Model and Notation (BPMN) - Version 2.0*. Object Management Group (OMG).

OMG (2011b). *Unified Modeling Language, Infrastructure, Version 2.4.1*. OMG.

OMG (2014). *Meta Object Facility (MOF) Core Specification, Version 2.4.2*.

Pardillo, J. (2010). A systematic review on the definition of uml profiles. In *Model Driven Engineering Languages and Systems*, pages 407–422. Springer.

Pfeiffer, D. and Gehlert, A. (2005). A framework for comparing conceptual models. In *Proceedings of the Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2005)*, pages 108–122. Citeseer.

Ralyté, J., Rolland, C., and Deneckère, R. (2004). Towards a meta-tool for change-centric method engineering: A typology of generic operators. In *Advanced Information Systems Engineering*, pages 202–218. Springer.

Scheer, A.-W. and Nüttgens, M. (2000). *ARIS architecture and reference models for business process management*. Springer.

Schuette, R. and Rotthowe, T. (1998). The guidelines of modeling–an approach to enhance the quality in information models. In *Conceptual Modeling–ER'98*, pages 240–254. Springer.

Sen, S., Moha, N., Baudry, B., and Jézéquel, J.-M. (2009). Meta-model pruning. In *Model Driven Engineering Languages and Systems*, pages 32–46. Springer.

Silingas, D. and Butleris, R. (2009). Towards customizing uml tools for enterprise architecture modeling. *Information Systems*, pages 25–27.

Stark, J. and Esswein, W. (2012). Rules from cognition for conceptual modelling. In *Conceptual Modeling*, pages 78–87. Springer.

Strahringer, S. (1996). *Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysenmethoden*. PhD thesis, TU Darmstadt.

Stroppi, L. J. R., Chiotti, O., and Villarreal, P. D. (2015). Defining the resource perspective in the development of processes-aware information systems. *Information and Software Technology*, pages 86–108.

Wand, Y. and Weber, R. (2002). Research commentary: information systems and conceptual modeling—a research agenda. *Information Systems Research*, 13(4):363–376.

zur Muehlen, M. and Recker, J. (2008). How much language is enough? theoretical and practical use of the business process modeling notation. In *Advanced information systems engineering*, pages 465–479.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS