How are Metamodels Specified in Practice? Empirical Insights and Recommendations

Completed Research

Dominik Bork
Research Group Knowledge Engineering
University of Vienna
dominik.bork@univie.ac.at

Dimitris Karagiannis
Research Group Knowledge Engineering
University of Vienna
dk@dke.univie.ac.at

Benedikt Pittl
Research Group Knowledge Engineering, University of Vienna
benedikt.pittl@univie.ac.at

Abstract

Conceptual modeling languages such as BPMN and UML are widely used in industry and academia. Such modeling languages are usually introduced in overarching specification documents maintained by standardization institutions. Being the primary - often even the single - source of information, such specifications are vital for modelers, researchers, and tool vendors. However, how to derive a coherent and comprehensive specification was never systematically analyzed. This paper reports on the analysis of 11 current modeling language specifications with a focus on how metamodels are specified. Identified metamodel specification techniques are discussed and their sample usage is illustrated. Thereby, individual strengths and weaknesses of each technique are discussed. The contribution of this paper is a foundation for increasing the consistency and expressive power of metamodel specifications, ultimately leading to an improved understanding and better utilization of modeling languages.

Introduction

Conceptual modeling languages such as the Business Process Modeling and Notation (BPMN) and the Unified Modeling Language (UML) are widely used in academia and industry. A single cooperation usually uses thousands of visual models (Rosemann, 2006) for systems analysis and design. Most conceptual modeling languages are introduced in overarching specifications. These specifications are therefore vital for: **modelers**, interested in understanding and applying a modeling language, **researchers**, aiming to evaluate and adapt modeling languages, e.g., to domain-specific purposes (Karagiannis et al., 2016), and **tool vendors**, interested in developing tools for a modeling language.

While the importance of modeling language specifications is obvious, to the best of our knowledge, structure, content, and specification techniques were never systematically analyzed. When creating a new modeling language specification, researchers are not guided by any best practice as existing specifications differ significantly. All this makes specifications difficult to comprehend. This paper shows that visual metamodel representations are a main pillar for specifying the syntax of most of today's modeling languages used in practice. The oldest UML specification which was released by the Object Management Group (OMG) - version 1.3 in 2000 - uses already a visual metamodel representation. Other modeling language specifications were reverting to visual metamodel representations later: e.g. for the BPMN since its version 2.0 release in 2011. While the importance of representing the syntax visually is obvious, other language aspects such as notation, constraints, serialization formats and execution semantics are introduced in specifications, too. Due to limitations of space, we will however focus on the modeling language specification by means of visual metamodels in the following.

The aim of this paper is to identify techniques for visually specifying metamodels. The contribution of this paper is of benefit for researchers, aiming to create a modeling language specification; for maintaining institutions, intending to improve existing specifications; and for modelers, interested in learning how to comprehend relevant information from overarching specifications. Thus, this paper establishes a first step toward improved and more comprehensive visual metamodel specifications for modeling languages. To

achieve this, current specifications of conceptual modeling languages are systematically analyzed with the aim to establish a foundation for metamodel specification techniques.

The remainder of the paper is structured as follows: In Section 2 foundations of metamodels and related work are introduced. Section 3 summarizes the research questions and the applied research methodology. The results of the analysis are then presented in Section 4. A discussion of the key findings and some recommendations for improving visual metamodel specifications are given in Section 5. The paper closes with concluding remarks and some future research directions in Section 6.

Background and Related Work

Terminological Foundation

As the scientific community uses different terms to refer to elements of metamodels, a terminology which will be used in the following is established now. The most important terms are visualized in Figure 1. All elements which occur in the metamodel are called *metaelements* or elements of the metamodel. Metaelements are either *object types* - represented by metaclasses - or *connector types*. Both types can be instantiated. Instances of metaclasses become objects in the model layer. Following the definition of Karagiannis and Kühn (2002), modeling methods are composed of *modeling language*, *modeling procedure*, and *mechanisms & algorithms*. Modeling languages are vital as they constitute a prerequisite for the latter two. Consequently, most specifications focus on the modeling language part, more precisely on the syntax of a modeling language - often referred in UML/OMG-related literature as abstract syntax. The syntax of a modeling language can be specified on different levels of formality. One of the most used techniques to formally specify the syntax is by using visual metamodel representations (Bork and Fill, 2014; Kleppe, 2008). In addition, a modeling language specification comprises notation - often referred in UML/OMG-related literature as concrete syntax - and semantics.

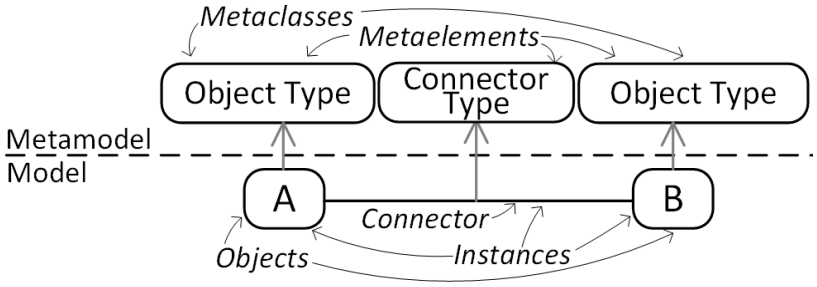

Figure 1. Terminological foundation metamodel and model elements.

Related Work

Recently, e.g. in (Kalnins and Barzdins, 2016; Cicirelli et al., 2016; Bork, 2015), visual metamodel representations were used to describe the syntax of modeling languages. The metamodels use different concepts such as *inheritance*, *aggregation* or *composition*, which effect the expressiveness of the specification. To the best of our knowledge, no scientific work exists which analyses the set of possible concepts of visual metamodel specification. Works that focus solely on modeling languages exist in the field of business process management. For example, a survey on business process standards is reported in (Ko et al., 2009). The authors considered all standards which belong to the business processes domain such as interchange formats or execution standards. In total, the survey comprised only two visual modeling languages - BPMN and UML. The survey moreover focused on the analysis of their modeling capabilities. A survey on business process validity modeling was published by Rosa et al. (2017). The survey (van der Aalst, 2013) analyzes business process management as management approach - the process languages itself are not analyzed. In Bork and Fill (2014), modeling language specifications are analyzed according to their formality, whereas Bork et al. (2018) focusses on analyzing visual expressiveness of conceptual modeling languages. Much work exists that targets the development of metamodels (e.g., Frank, 2010). Moreover, tool vendors often publish their meta-metamodels. In contrast to the aforementioned, this paper focuses on visual specification techniques for metamodels. It is not within the scope of this research to analyze constituents, quality, or development of metamodels.

Scientific publications which analyze expressiveness and consistency of selected modeling languages are also available. For example, in (Allaki et al., 2017) inconsistencies of the UML were analyzed without analyzing metamodels. In Henderson-Sellers and Ralyté (2010), selected metamodels used for *situational method engineering* were analyzed. However, general concepts of metamodel representation techniques were not analyzed. Several works have been published, focusing on the quality of metamodels, e.g., by proposing quality metrics (Ma et al., 2013; Williams et al., 2013; di Rocco et al., 2014). Hinkel et al. (2016), provides an empirical investigation on how students evaluated metamodels.

All of the mentioned works consider the analysis of the modeling language but do not focus on the metamodels. So far, no scientific work provides guidance and/or best practices in how to specify modeling languages by means of visual metamodel specification techniques. This is a serious research gap, as such specifications are the primary - often the single - source of knowledge for heterogeneous stakeholders.

Research Questions and Research Methodology

For conducting the analysis, we followed a three-phase research method (Kitchenham and Brereton, 2013) comprising a *planning phase*, a *conduction phase*, and a *result phase*. In the planning phase the research questions as well as the inclusion/exclusion criteria are defined. The conduction phase then describes the execution of the analysis. Finally, the results are presented in the result phase. The planning and the conduction phase are described in the following two subsections. The rest of the paper then comprehensively presents and discusses the results.

Planning the Analysis

The term conceptual modeling language (cf. Karagiannis and Kühn, 2002) is applied for languages which allow the creation of diagrammatic models (Buchmann and Karagiannis, 2016) - called visual models in this paper. The goal of the analysis is to answer the following research question: *How are metamodels of conceptual modeling languages visually specified in practice?* Practice, refers to modeling languages which are heavily used in industry. A systematic analysis of modeling language specifications is required to answer this research question. We considered specification documents which fulfill the following inclusion criteria: Document is declared as *specification*, *definition* or *standard*; Document describes a *conceptual modeling language*; Document describes a *metamodel*; Document is *freely accessible*.

Our approach for finding relevant specifications was twofold: First, institutions which specify modeling languages are well known: OMG and OpenGroup. We systematically analyzed the specifications published on their websites. Next, we conducted a systematic keyword search on www.google.com with the query (*Modeling Language*) AND (*Specification OR Definition OR Standard OR Description OR Documentation*). The search was conducted in June 2017. For each combination of terms, we analyzed the first ten pages of the search result. For our analysis, scientific databases such as DBLP or Google Scholar were not focused, because modeling language specifications are usually published by maintaining institutions. We explicitly excluded scientific publications that only propose incomplete specifications or extensions of existing modeling languages (e.g. UML profiles). Lastly, we excluded specifications that were not written in English and/or not revised since 01.01.2012. These exclusion criteria were established for ensuring that only complete and current modeling languages are considered.

Conducting the Analysis

The results of the Google keyword search referred several times to pages such as Rosetta Standards[1] or IDEF[2] which enforce a registration before one can access the specifications. If the registration was free of charge we created an account. The Climate Science Modelling Language (CSML)[3] was not accessible and therefore not analyzed. The Object Process Methodology (OPM) specification is not freely available. However, a working draft version is accessible which we used for our survey.

[1] https://resources.gs1us.org/rosettanet, last accessed: 26.02.2018

[2] http://www.idef.com/, last accessed: 24.02.2018

[3] http://csml.badc.rl.ac.uk/, last accessed: 22.02.2018

All specifications that met all inclusion criteria were classified as being relevant. Heading and introduction sections were read in order to ensure that the inclusion criteria IC 1 and IC 2 are fulfilled. Two specification documents were found for the BPMN: one from the OMG and one from the ISO. Similarly, there exists an UML specification from ISO and one from OMG. In both cases, the specification documents from the OMG were used. The Decision Modeling Notation (DMN) introduces two languages: the Decision Requirements Diagram (DRD) which has a graph-based structure and decision tables which have no such structure. Thus, only the DRD was analyzed. In total, 17 relevant specifications were found. On the websites of OMG and OpenGroup *further* 6 relevant specifications have been identified.

In a second step, the 23 specifications have been systematically evaluated along the exclusion criteria by reading the introduction section and by cross-reading the following sections. Furthermore, the table of contents has been considered to identify the scope of the specification. This evaluation step followed a peer-review process by the authors, thereby classifying the specifications using three categories: *complying to the exclusion criteria*, *not complying to the exclusion criteria* and *further evaluation needed*. For example, the System Structure Modeling Language (S2ML) specification required further evaluation as it has a strong focus on the introduced textual language - not on the conceptual modeling language. Also the business motivation model (BMM) specification required further evaluation because of the limited notation. The specifications belonging to this category were evaluated independently again and afterwards discussed by the authors to come to a decision. Finally, 11 specifications which comply with the search criteria have been identified (see Table 1).

Metamodel Specification Techniques

By analyzing the 11 modeling language specifications with a focus on how the metamodels have been specified, six visual metamodel specification techniques have been classified which will be discussed and their sample usage illustrated in the following.

Slicing Metamodels

Metamodels are usually large and cannot be represented in one single figure/diagram. Thus, specifications such as UML and ArchiMate use a representation technique which will be referred to in the following as *slicing*. The metamodel is decomposed into multiple slices. Each slice has at least one element which is part of another slice enabling the complete metamodel to be re-constructed by merging the slices. An example is shown in Figure 2, where the complete metamodel is decomposed into three slices. Each slice has one element which is part of another slice - classes A and D.

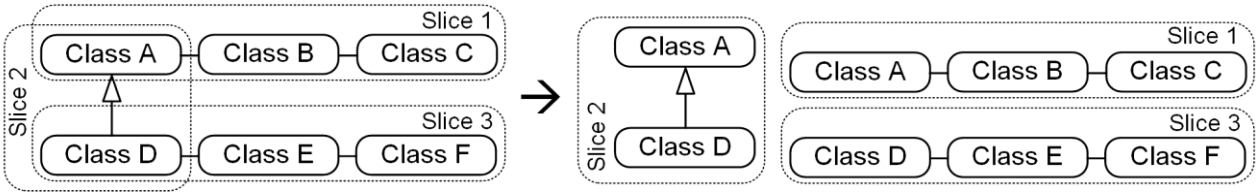

Figure 2. Metamodel decomposition into slices.

A further distinction between *redundant slicing* and *non-redundant slicing* is introduced. In the example depicted in Figure 2, the non-redundant slicing approach is used as *exactly one element* of each slice is used in another slice. By contrast, redundant slicing refers to cases in which several elements of one slice are part of another slice. All analyzed specifications use the redundant slicing approach. An example from BPMN (BPMN, 2018) is depicted in Figure 3. In both slices, attributes and relationships of the elements *BaseElement* and *Documentation* are present.

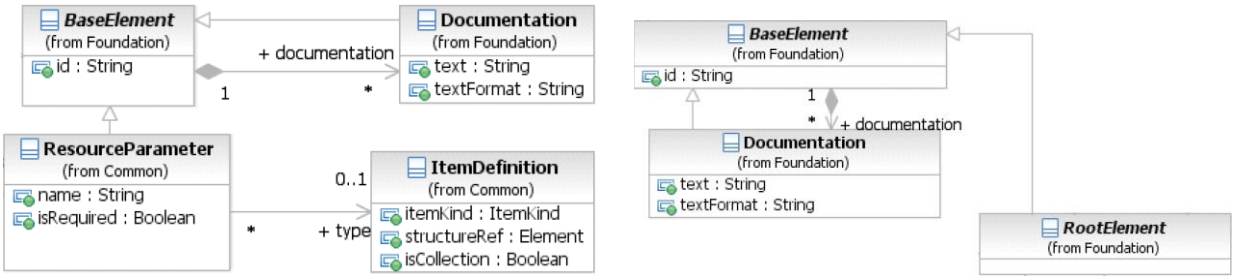

Figure 3. Example slides from the BPMN.

Referencing Metamodels

The slicing metamodel specification technique comes with a serious issue. If elements of the metamodel are present in multiple slices, redundancy is inevitable. Moreover, if at all places, all aspects of the element are specified, i.e., not only the name but also attributes and relationships to other elements, comprehension by humans is impeded and inconsistencies after metamodel revisions are likely.

In order to avoid these issues, six out of the 11 analyzed specifications such as for URN (URN, 2018) use special reference elements - see Table 1. Such reference elements contain only the name of the metamodel element while omitting additional information such as attributes. Thus, any element is only once specified in detail but can be referenced multiple times throughout the metamodel specification. Such reference elements reduce redundancy. Consequently, reference metamodel specifications can be considered a specialized form of slicing metamodels.

Generic Metamodels

Generic metamodels do not describe the syntax of the modeling language - their only purpose is to foster understanding of *the structure* of the metamodel by providing generic concepts. ArchiMate (ArchiMate, 2018) extensively uses generic metamodels as an example visualized in Figure 4 shows.

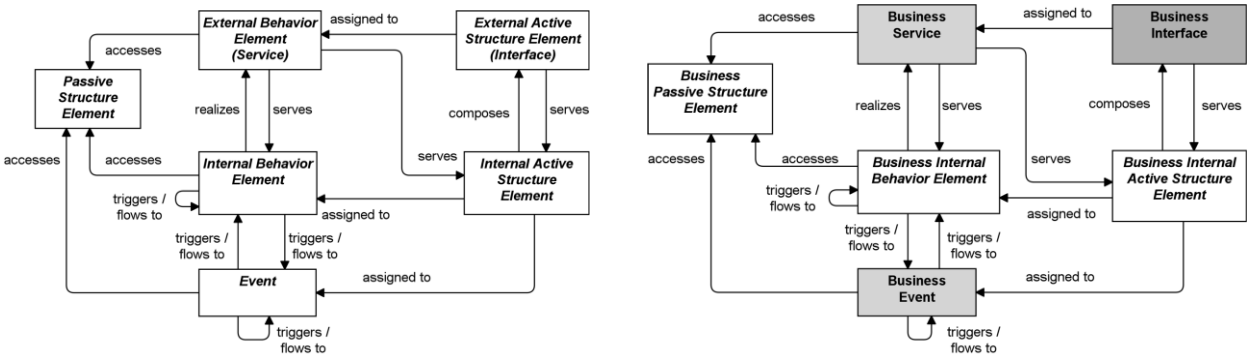

Figure 4. Generic metamodel (left) and business layer metamodel (right) of ArichMate.

Figure 4 (left) depicts the generic metamodel with abstract metaclasses *Internal Behavior Element* and *Passive Structure Element* while Figure 4 (right) shows the metamodel of the business layer, mixing real metaclasses (e.g., *Business Service*) with semantic containers of the generic metamodel (e.g., *Business Internal Behavior Element*). The latter metamodel uses specifications of the classes depicted in former.

Notation-aware Metamodels

Usually, metamodels describe the syntax of a modeling language without considering the notational aspects. However, the analysis revealed *notation-aware metamodels* which combine the specification of syntax with notation (abstract with concrete syntax in UML terminology). Such metamodels distinguish between conventional metaclasses and specific metaclasses for the notational aspects. The latter metaclasses have no semantics, they contain attributes that are solely relevant for the notation.

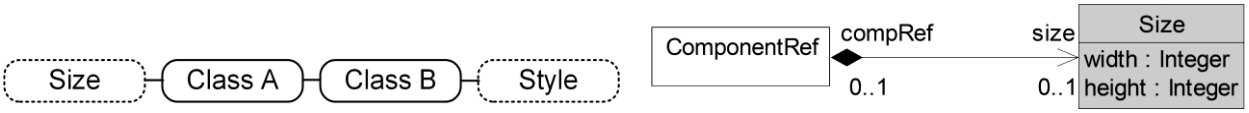

Figure 5. Generic notation-aware metamodels (left) and sample in URN (right).

Figure 5 (left) shows a generic structure of a notation-aware metamodel. *Classes A* and *B* are conventional metaclasses whereas the classes *Size* and *Style* are specific metaclasses for the notation. For example, the style class might have a boolean attribute which indicates if the metaclass is filled with a color or not. The URN specification (URN, 2018) uses such notation-aware metamodels. Figure 5 (right) shows how the metaclass *ComponentRef* is being connected to the notation-specific metaclass *Size*. It needs to be stated, that the URN specification was the only one out of the 11 analyzed that uses such notation-aware metamodels. As such metamodels are however commonly used e.g., in scientific publications that cover metamodels (Ferstl et al., 2016), we consider this technique as being noteworthy.

Matrix Metamodels

Four out of the 11 analyzed specifications use matrices instead of or in addition to visual metamodel representations - see Table 1. Such matrices show on both axes the metaclasses of the modeling language which can be instantiated by a modeler and the allowed connections between them. Figure 6 (left) shows an excerpt of a matrix specification of LML (LML, 2018). *Action*, *Artifacts*, and *Asset* are metaclasses. *Resource* is a subclasses - indicated by the brackets. The connector types which are surrounded by the brackets - i.e., *consumes*, *produces* and *seizes* - are only valid to be used between the sublcasses. Thus, instances of the *Resource* metaclass can be connected to instances of the *Action* metaclass by an instance of the *consumes* connector type.

	Artifact	Asset (Resource)
Action	references	(consumes) performed by (produces) (seizes)

Entity	Parent Entity	Child Entities	Description
Action	None	None	An **Action** entity generates effects and may have pre-conditions before it can be executed. This **Action** can include transforming inputs into outputs. Examples: Process, Discover, Calculate.

Attribute	Data Type	Description
duration	Number	*duration* represents the period of time this **Action** occurs.

Relationship	Inverse	Target Entity	Description
consumes	***consumed by***	**Resource**	***consumes*** identifies the **Resource** that this **Action** uses. After this **Action** is completed the amount consumed is not returned to the **Resource**.

Attribute	Data Type	Description
amount	Number	amount represents how much of the resource is ***consumed by*** the **Action**. Units are relative to the units selected for the **Resource**.

Figure 6. Matrix (left) and tabular (right) specification technique usage in LML

Tabular Metamodels

This technique also represents the metamodel in a textual way. For a specific metaclass, the table lists all attributes with data types, the allowed connector types, and optional inheritance relationships. The LML specification (LML, 2018) uses such tabular specifications. An example is depicted in Figure 6 (right), showing an excerpt of the specification of the metaclass *Action*. Eight out of the 11 analyzed specifications use a simplified table. Most of them use the table in addition to the visual metamodel to specify the semantics of the metaclasses by means of their attributes - see Table 1.

Discussion

In the following, a comprehensive discussion of the key observations is performed which then leads to a critical reflection on strengths and weaknesses of each of the identified specification techniques and ultimately to some recommendations for improving modeling language specifications in the future.

Observations

Table 1 summarizes the specification techniques used by the analyzed modeling languages. Nine specifications use visual metamodel representations, eight specifications use tables and four specifications use matrices. Out of the eight analyzed specifications that use slicing metamodels, six used referencing elements to indicate the relationships between the metamodel slices. Generic metamodels were used by ArchiMate extensively, whereas notation-aware metamodels have been used by URN.

Modeling Standard	Maintaining Institution	Metamodel Visualization Technique						
		Visual Metamodel	Slicing Metamodel	Referencing Metamodel	Generic Metamodel	Notation-aware Metamodel	Matrix	Table
ArchiMate 3.0.1	OpenGroup	Y	Y	N	Y	N	Y	Y
BPMN 2.02	OMG	Y	Y	Y	N	N	Y	Y
CMMN 1.1	OMG	Y	Y	Y	N	N	N	Y
DMN 1.1	OMG	Y	Y	Y	N	N	Y	Y
IFML 1.0	OMG	Y	Y	N	N	N	N	Y
LML 1.1	LML	N	-	-	-	-	Y	Y
OPM 522	ISO	Y	N	N	N	N	N	Y
S2ML 1.0	OpenAltaRica	N	-	-	-	-	N	N
UML 2.5	OMG	Y	Y	Y	N	N	N	Y
URN Z.151	ITU-T	Y	Y	Y	N	Y	N	N
VDML 1.5	OMG	Y	Y	Y	N	N	N	N

Table 1. Metamodel specification techniques used in practice

Interestingly, even for those specifications maintained by the same institutions, i.e., the Object Management Group (OMG), and/or even sharing the same meta-metamodel, e.g., IFML, UML, and VDML, significant differences have been identified. These differences not only reflect the metamodel specification techniques but also the extent to which syntax, semantics, and notation of a modeling language are specified (not further discussed in this paper).

The slicing approach is widely used in the analyzed specifications. Only the OPM specification, which introduces a single visual metamodel in the specification called metamodel overview, does not use slicing. This is of course related to the size of the metamodels. For most modeling languages, the visual representation of the complete metamodel is just not feasible. Surprisingly, none of the specifications used a tree based representation for the metamodels such as employed by Ecore-based metamodels[4].

As Table 1 shows, specifications usually use several metamodel specification techniques in combination whereby matrices and tables are used to summarize the syntax introduced by visual metamodels - the only exceptions are the LML and the OPM. In general, all of the specification techniques have their strengths and weaknesses. This is why the analyzed specifications all use a combination of those techniques. There exists no 'one size fits it all' specification technique.

[4] Ecore is the meta-metamodel used within the Eclipse EMF project.

Critical Reflection and Recommendations

In the following, a critical reflection on the core strengths and weaknesses of the six identified metamodel specification techniques is given. Based on this reflection, practical and theoretical implications are derived.

Slicing Metamodels are commonly used throughout the analyzed specifications, because of the sheer size of the metamodels. In most cases, it is just not feasible to visualize the complete metamodel on one figure/page in a comprehensible form. Hence, decomposition is inevitable. Redundant and non-redundant slicing foster understanding of how the slices fit together in order to cognitively re-construct the overarching metamodel. Due to the inherent redundancy, consistency needs to be considered.

Referencing Metamodels moderate the redundancy issue of slicing metamodels by limiting the slice to only one element which is not specified in detail but only as a reference. In this regard, referential integrity needs to be considered, i.e., changes in the referenced element may have an impact on the referencing element.

Generic Metamodels enable semantic de-/composition of overarching metamodels. In this regard, they contribute to a richer decomposition of complex metamodels into slices. At the same time, they might also increase the complexity as new concepts are introduced to metamodels that are actually not part of the modeling language.

Notation-aware Metamodels enable the metamodels to not only cover the syntax (abstract syntax) but also the notation (concrete syntax) of metaelements. A single point of information can be realized while at the same time diluting the expressive power of metamodels. A clear separation between the solid syntactic metaelements and their possibly interchangable graphical representation is advisable.

The widely used **Matrix Metamodels** have several drawbacks. For example, the representation of class hierarchies is limited, multiplicities or cardinalities cannot be represented properly, and connector types such as composition and non-composition cannot be represented intuitively. On the other side, matrix representations enable to gain an immediate overview of how metaclasses can be related to each other by means of connector types. Thus, matrices may need to be accompanied by another technique to create a comprehensive specification.

Tabular Metamodels usually create redundancy. For example, connector types including their attributes which are used for connecting several metaclasses have to be re-introduced for each class. However, tabular metamodels are powerful when used to specify the attributes of metaclasses.

Based on these results, some practical and theoretical implications for improving modeling language specifications in the future are derived. As modeling languages are under continuous revision and extension, we believe this research make a meaningful contribution to create better specifications.

Completeness. The analysis revealed not only the heterogeneous set of used techniques but also a lack of completeness. Connectors and/or attributes are not specified precisely. This is a serious deficit as only complete specifications enable proper understanding, utilization, and tooling. We recommend to check whether for each metaelement syntax, semantics, and notation are specified completely.

Consistency. The analysis revealed several weaknesses e.g., with regards to the inconsistent usage of metaclasses, connector types, and alternative notations. Moreover, specification techniques are not used consistently, impeding comprehension the specification. We recommend to consistently use one/a set of specification techniques throughout the whole specification and to decide, which specification technique to employ for which facets of the modeling language.

Separation of Concerns. Based on the analysis, we can suggest that it is best to separate syntax, semantics, and notation of a modeling language and specify them using the most suitable specification technique. We recommend to use slicing and referencing metamodels as a powerful and compact technique for specifying syntactic aspects, whereas tables are good for specifying the semantics and notation of a modeling language.

Technique Mix. All investigated specifications use two or more techniques throughout the document. When done in a meaningful way, this enables the combination of the positive aspects of each of them while comprising a comprehensive modeling language specification. We recommend to use a mix of

specification techniques in order to utilize the respective strengths of the techniques in order to specify certain modeling language aspects.

Preamble. What is important is that the reader of the specification is aware of the structure and techniques used within the specification. This contributes to a better and faster comprehension of the actual modeling language. We recommend to provide an extended preamble of the specifications where not only the used terminology but also the applied specification techniques are introduced.

From a theoretical standpoint, this research establishes a taxonomy of different currently used visual metamodel specification techniques. Based on these results, researchers are enabled to create a theory on e.g., the perceived usefulness or the ease of use of different specification techniques. The influence of different techniques on comprehension and learnability of modeling languages needs to be researched in the future. The paper at hand establishes a mandatory foundation for such studies.

Conclusion

To the best of our knowledge no research exists that provides some guidance in how to specify modeling languages. Due to the wide adoption of such language specifications both in academia and industry, this is a serious research gap. This paper contributes bridging that gap by systematically analyzing 11 current specifications with a focus on the visual metamodel specification techniques. After a thorough introduction of the identified techniques with samples from the specifications and a critical reflection, the paper provides practical and theoretical implications for improving metamodel specifications. This research is of great value for three stakeholders: i) *researchers*, aiming to create a specification for a conceptual modeling language; *maintaining institutions*, intending to improve existing specifications; and *modelers*, interested in learning how to comprehend relevant information out of overarching specification documents.

In future research we aim to broaden the scope of the analysis in order to incorporate further modeling language aspects like notation and semantics. First results indicate, that also for those aspects heterogeneous specification techniques can be identified. Moreover, we aim to apply the findings to draft concrete improvements for existing specifications. We are aware of the limitations of the current study. It is therefore our goal to empirically underpin our recommendations by interviewing users on how they perceive different modeling language specification techniques. Thus, this research establishes the foundation for theory-development with regards to perceived usefulness and ease of use of metamodel specification techniques.

Acknowledgement

Part of this research has been funded through the South Africa/Austria Joint Scientific and Technological Cooperation program with the project number ZA 11/2017.

REFERENCES

Allaki, D., Dahchour, M., and En-Nouaary, A. 2017. “Managing inconsistencies in UML models: A systematic literature review,” *JSW* (12:6), pp. 454-471.

ArchiMate 2018. “ArchiMate,” The Open Group, https://www2.opengroup.org/ogsys/catalog/C162, Accessed on 2018-02-22.

BPMN 2018. “Business process model and notation,” OMG, http://www.omg.org/spec/BPMN/2.0.2/PDF/, Accessed: 2018-02-22.

Bork, D., and Fill, H.G. 2014. “Formal aspects of enterprise modeling methods: a comparison framework,” in *2014 47th Hawaii International Conference on System Sciences,* IEEE, pp. 3400-3409.

Bork, D. 2015. “Using conceptual modeling for designing multi-view modeling tools,” in *21st Americas Conference on Information Systems*, AMCIS 2015, Puerto Rico.

Bork, D., Karagiannis, D., and Pittl, B. 2018. “Systematic Analysis and Evaluation of Visual Conceptual Modeling Language Notations,” in *IEEE 12th International Conference on Research Challenges in Information Science (RCIS'2018)*, 29-31 May 2018, Nantes, France, in press.

Buchmann, R.A., and Karagiannis, D. 2016. „Enriching linked data with semantics from domain-specific diagrammatic models,” *Business & Information Systems Engineering* (58:5), pp. 341-353.

Cicirelli, F., Fortino, G., Guerrieri, A., Spezzano, G., and Vinci, A. 2016. "Metamodeling of smart environments: from design to implementation," *Advanced Engineering Informatics*, (33), pp. 274-284.

CMMN 2018. "Case management model and notation," OMG, http://www.omg.org/spec/CMMN/1.1/PDF/, Accessed: 2018-02-23.

Di Rocco, J., Di Ruscio, D., Iovino, L., and Pierantonio, A. 2014. "Mining metrics for understanding metamodel characteristics," in *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, ACM, pp. 55–60.

DMN 2018. "Decision modeling notation," OMG, http://www.omg.org/spec/DMN/1.1/PDF/, Accessed: 2018-02-22.

Ferstl, O.K., Sinz, E.J., and Bork, D. 2016. "Tool Support for the Semantic Object Model," in *Domain-specific Conceptual Modeling*, Springer International Publishing, Cham, pp. 291–310.

Henderson-Sellers, B., and Ralyté, J. 2010. "Situational method engineering: State-of-the-art review," *J. UCS* 16(3), pp. 424-478.

Hinkel, G., Kramer, M., Burger, E., Strittmatter, M., and Happe, L. 2016. "An empirical study on the perception of metamodel quality," in *Model-Driven Engineering and Software Development (MODELSWARD), 4th International Conference on*, IEEE, pp. 145-152.

IFML 2018. "Interaction flow modeling language," OMG, http://www.omg.org/spec/IFML/1.0/PDF/, Accessed: 2018-02-23.

Kalnins, A., and Barzdins, J. 2016. "Metamodel specialization for graphical modeling language support," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. MODELS '16, New York, NY, USA, ACM, pp. 103-112.

Karagiannis, D., Mayr, H.C., and Mylopoulos, J. 2016. *Domain-Specific Conceptual Modeling, Concepts, Methods and Tools*, Springer.

Karagiannis, D., and Kühn, H. 2002. "Metamodelling platforms," in *E-Commerce and Web Technologies, Third International Conference, EC-Web 2002*, Aix-en-Provence, France, p. 182.

Kitchenham, B., and Brereton, P. 2013. "A systematic review of systematic review process research in software engineering," *Information & Software Technology* (55:12), pp. 2049-2075.

Kleppe, A. 2008. *Software language engineering: creating domain-specific languages using metamodels*, Pearson Education.

Ko, R.K.L., Lee, S.S.G., and Lee, E.W. 2009. "Business process management (BPM) standards: a survey," *Business Process Management Journal* (15:5), pp. 744-791.

LML 2018. "Lifecycle modeling language," LML, http://www.lifecyclemodeling.org/specification/, Accessed: 2018-02-23.

Ma, Z., He, X., and Liu, C. 2013. "Assessing the quality of metamodels," *Frontiers of Computer Science* (7:4), pp. 558-570.

OPM 2018. "Object process methodology;" ISO, https://www.iso.org/standard/62274.html, Accessed: 2018-02-22.

Rosa, M.L., Van Der Aalst, W.M., Dumas, M., and Milani, F.P. 2017. "Business process variability modeling: a survey," *ACM Computing Surveys* (50:1), p. 2.

Rosemann, M. 2006. "Potential pitfalls of process modeling: part A," *Business Process Management Journal* (12:2), pp. 249-254.

S2ML 2018. "System structure modeling language,", https://hal.archives-ouvertes.fr/hal-01234903/document, Accessed: 2018-02-22.

UML 2018. "Unified modeling language specification," OMG, http://www.omg.org/spec/UML/2.5/PDF/, Accessed: 2018-02-21.

URN 2018. "User requirements notation," ITU-T, https://www.itu.int/rec/T-REC-Z.151-201210-I/en, Accessed: 2018-02-23.

Van Der Aalst, W.M. 2013. "Business process management: a comprehensive survey," *ISRN Software Engineering*.

VDML 2018. "Value delivery modeling language," OMG, http://www.omg.org/spec/VDML/1.0/PDF/, Accessed: 2018-02-23.

Williams, J.R., Zolotas, A., Matragkas, N.D., Rose, L.M., Kolovos, D.S., Paige, R.F., and Polack, F.A. 2013. "What do metamodels really look like?," in Eessmod@ Models 1078, pp. 55-60.