A survey of modeling language specification techniques

Dominik Bork, Dimitris Karagiannis and Benedikt Pittl

A Survey of Modeling Language Specification Techniques

Dominik Bork[a,*], Dimitris Karagiannis[a] and Benedikt Pittl[a]

[a]*University of Vienna, Faculty of Computer Science, Waehringer Street 29, 1090 Vienna*

ARTICLE INFO

Keywords:
Modeling Language
Specification
Metamodel
Evaluation
Standards

ABSTRACT

Visual modeling languages such as the Business Process Model and Notation and the Unified Modeling Language are widely used in industry and academia for the analysis and design of information systems. Such modeling languages are usually introduced in overarching specifications which are maintained by standardization institutions such as the Object Management Group or the Open Group. Being the primary - often the single - source of information, such specifications are of paramount importance for modelers, researchers, and tool vendors. However, structure, content, and specification techniques of such documents have never been systematically analyzed. This paper addresses this gap by reporting on a Systematic Literature Review aimed to analyze published standard modeling language specifications. In total, eleven specifications were found and comprehensively analyzed. The survey reveals heterogeneity in: i) the modeling language concepts being specified, and ii) the techniques being employed for the specification of these concepts. The identified specification techniques are analyzed and presented by referring to their utilization in the specifications. This survey provides a foundation for research aiming to increase consistency and improve comprehensiveness of information systems modeling languages.

1. Introduction

Visual modeling languages are widely used in academia [26] and industry [51]. A single enterprise usually uses thousands of visual models [70]. These modeling languages are commonly introduced in overarching specification documents. Such specifications are therefore vital for: (i) modelers, aiming to comprehend a modeling language, (ii) researchers, aiming to evaluate and adapt a modeling language, and (iii) tool vendors, aiming to develop a modeling tool (cf. [21]). While the importance of such specifications is unquestioned, to the best of our knowledge, structure, content, and specification techniques have never been systematically analyzed. Some prominent modeling languages such as Entity-Relationship Diagram (ER) and Event-driven Process-Chains (EPC) neither have a proper specification nor a maintaining organization - see [77] and [67, 27] for interesting discussions. Similar discussions on Use Cases are reported in [79]. In 1990, a standard for ER was proposed [74] which was never adopted in practice [72]. Consequently, inconsistent definitions for ER-concepts such as *Entity* exist and different notations are used [77]. Moreover, tool vendors can adapt or researchers can extend a modeling language in a way that meets their specific objectives. This results in compatibility issues and aggravates comprehension. Indeed, the absence of complete and consistent specifications may lead to a limited usage and acceptance of a modeling language as the example of the EPC shows [67, 27, 24]. Now, the business process community puts remarkable effort into the development of a specification for the EPC [67, 27, 36, 41], similarly the Object Management Group (OMG) is increasingly concerned with improving their specifications, e.g. with respect to formal aspects [29, 14, 76].

Researchers, aiming to create a new or extend a given modeling language, e.g., [37, 3], heavily rely on the modeling language specification. Furthermore, existing specifications differ significantly in the techniques they use, the structure they employ, and the formality [11] - even in those specifications maintained by the same institution and using the same meta-metamodel. All this makes the comprehension of a modeling language specification a cumbersome and error-prone task. Based on our previous works [12, 13] this survey shows that visual metamodels are a main pillar for specifying the syntax of most of today's modeling languages. A visual metamodel, as referred to in the following, specifies modeling language aspects by graphical means, i.e., not purely textual. The first specification of the Unified Modeling Language (UML) which was released by the OMG - version 1.3 in 2000[1] - already uses visual metamodels.

*Corresponding author

dominik.bork@univie.ac.at (D. Bork)
ORCID(s): 0000-0001-8259-2297 (D. Bork)

[1]http://www.omg.org/spec/UML/, last accessed: 2019-01-22

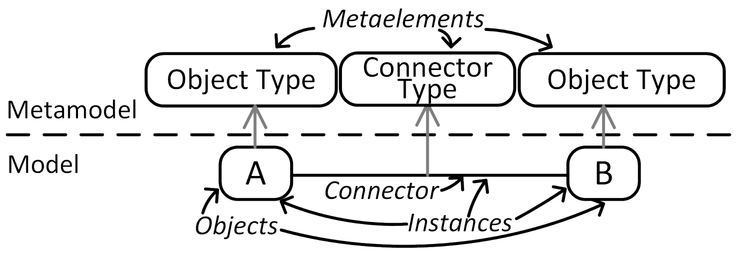


Figure 1: Used terms for metamodel and model elements

With the release of version 2.0 in 2011, the Business Process Model and Notation (BPMN) used visual metamodels for the first time. Besides the focus on visual metamodels, other language aspects such as notation, constraints, serialization formats and execution semantics are introduced in specifications, too.

This research surveys existing modeling language specifications with the objective to identify specification techniques for all aspects of a conceptual modeling language [56, 39]. The contribution of this paper is of benefit for researchers and practitioners aiming to create a specification for a modeling language (e.g., a domain-specific modeling language [39]), for maintaining institutions aiming to improve existing specifications, and for modelers who want to learn how to comprehend relevant information from overarching specification documents.

The remainder of this paper is structured as follows: In section 2 foundations and related work are introduced. Section 3 summarizes the research questions and the research methodology employed for conducting the survey. The remaining sections present the results of the survey: Specification techniques of visual metamodels are analyzed in section 4. Techniques for the specification of connector types in visual metamodels are analyzed in section 5 followed by section 6, where the specification of further metamodel concepts as proposed by Kern et al. [43] is surveyed. This section concludes with some further observations, e.g., with respect to the specification of constraints. Techniques for the specification of the modeling language notation are analyzed and the given notations are evaluated in section 7. The specification of interoperability and language extension aspects is analyzed in section 8 followed by a discussion on the validity of the survey in section 9. This paper closes with conclusions and some future research directions in section 10.

2. Background and Related Work

2.1. Terminological Foundation

As the scientific community uses different terms to refer to elements of metamodels a terminology which will be used throughout this survey needs to be established first. The most important terms are visualized in figure 1. All elements which occur in the metamodel layer are called *metaelements* or elements of the metamodel. A metaelement is either an *object type* or a *connector type*. Both types can be instantiated whereas instances of an object type become an *object* and instances of a connector type become a *connector* in the model layer.

Following the definition proposed in [39], modeling methods are composed of a *modeling language*, a *modeling procedure*, and *mechanisms & algorithms*. The modeling language is vital as it constitutes a prerequisite for the latter two. A modeling language specification comprises syntax (also referred to in the Unified Modeling Language (UML) literature as *abstract syntax*), semantics, and notation (also referred to in the UML literature as *concrete syntax*). However, most specifications focus on the syntax - which can be formally described using visual metamodels [11] - while omitting to formally specify the notation and particularly the semantics. This survey consequently focuses on visual metamodel specification techniques while also considering the specification of the notation.

2.2. Related Work

Analyses that focus solely on modeling languages exist in the field of business process management. For example, a survey on business process standards was conducted in [48]. In this survey, the authors did consider standards belonging to the business processes domain such as interchange formats or execution standards. The survey focused on the analysis of their modeling capabilities. Recently, a survey on business process validity modeling was published in [69]. The survey [80] analyzed business process management as management approach - the process languages themselves were not analyzed. An analysis of six enterprise modeling languages with a focus on different levels of formality in the specification techniques is reported in [11]. Related efforts to formalize [22] and to identify reusable modeling language abstractions [17] have been proposed recently.

Scientific publications which analyze the expressiveness and consistency of models [20] and selected modeling languages are also available. For example, in [1] the consistency of the UML was analyzed, but without analyzing metamodels. Paige et al. [65] analyzed two specification formalisms for metamodels in the domain of multi-view modeling. In [32], selected metamodels used for *situational method engineering* were analyzed. However, general concepts of metamodels as well as specification techniques were not focused on. An analysis of the semantics of Business Process Model and Notation (BPMN) that also identified deficiencies of its specification is reported in [21].

Several works focus on the quality of metamodels, e.g., by proposing metrics [8, 28, 52, 81, 19]. In [33], the authors empirically investigate how participants designed metamodels for the same application domain. After analyzing the metamodels in a peer review procedure, the authors claimed that *the perceived quality was mainly driven by the metamodels completeness, correctness and modularity* [33, p. 145]. In [53], a survey of formalisms for describing visual modeling languages is presented, comprising e.g. string grammars and attributed multiset grammars - metamodels where not analyzed. A classification of visual models along geometric-based classes, connection-based classes and hybrid classes was introduced in [16]. In [57] modeling languages realized with the Eclipse Modeling Framework were analyzed with a focus on metamodel quality metrics.

Metamodels introduced in research and practice use different concepts such as *inheritance*, *aggregation*, or *composition*. To the best of our knowledge, neither exists a scientific analysis regarding effectiveness and expressiveness of modeling language specifications, nor an analyses of the structure of and the techniques employed in such specifications. Existing works analyze one or a set of metamodels such as [49]. This is a serious research gap, as such specifications are the primary source for researchers, students, and practitioners aiming to comprehend, apply, or extend a given language - or to create a new one (cf. [51]).

3. Research Questions and Research Methodology

For conducting this survey, a three-step research method [44, 46] has been followed and the guidelines for conducting systematic reviews as proposed in [45] and applied in [68] have been respected. The research method comprises a *planning phase*, a *conduction phase*, and a *result phase*. Planning and conduction phase are described in the following two subsections. The rest of this paper then comprehensively discusses the results.

3.1. Planning the Survey

The term visual modeling language (cf. [39]) is applied to languages which allow the creation of diagrammatic models [15] - called visual models in this paper. The goal of this survey is to identify how visual modeling languages are specified in practice. Practice, in the context of this survey, refers to modeling languages which are heavily used in industry - and therefore specified by international institutions such as the Object Management Group (OMG). This survey answers the following research questions with results described in the referenced sections.

- RQ 1: Which techniques are used to specify a modeling language? → see sections 4, 5, 6, 7, 8
- RQ 1.1: How are visual metamodels structured? → see section 4, 5
- RQ 1.2: What concepts [43] are used in metamodel specifications? → see section 6
- RQ 1.3: Which techniques are used to specify a modeling language notation? → see section 7
- RQ 2: How are language interoperability and extension specified? → see section 8

To answer these research questions we surveyed and systematically analyzed existing modeling language specifications. We considered specifications which fulfill the following inclusion criteria:

- IC 1: Document is declared as *specification*, *definition*, or *standard*
- IC 2: Document describes a visual modeling language
- IC 3: Document is freely accessible

Our approach for finding such specifications was twofold: (i) Institutions which specify modeling languages are well known: e.g., OMG and OpenGroup. We systematically analyzed the specifications published on their websites[2].

[2]http://www.omg.org/spec/, http://publications.opengroup.org, last accessed: 2019-04-11

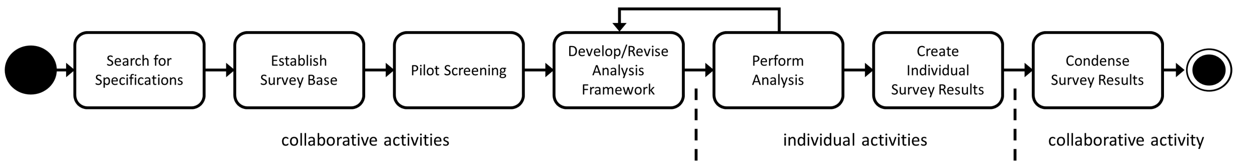


Figure 2: Individual and collaborative activities performed throughout the survey process

(ii) In addition, we conducted a systematic search on `www.google.com` with the keywords *Modeling Language Specification*, *Modeling Language Definition*, *Modeling Language Standard*, *Modeling Language Description* and *Modeling Language Documentation*. For each search term, we analyzed the first ten pages of the search results.

For our survey, scientific databases such as DBLP or Google Scholar were not analyzed because modeling language specifications are usually published by the maintaining institutions. We explicitly avoided to take into account scientific publications as they cover usually only excerpts of a modeling language for which neither tool support nor adoption by modelers can be expected for granted. We also excluded publications that solely discuss language extensions, e.g., Unified Modeling Language (UML) profiles. The complete set of exclusion criteria was as follows:

- EC 1: Document covers only an extension of another language specification
- EC 2: Document is incomplete: syntax, semantics, or notation is missing
- EC 3: Document was published before 01.01.2012
- EC 4: Document is not in English

3.2. Conducting the Survey

Figure 2 illustrates the survey process. The process is separated into individual and collaborative activities. All authors of this paper were involved in conducting the survey. In a collaborative effort, first all authors jointly conducted the search in order to establish the survey base. Afterwards a pilot screening was conducted in order to develop a fist analysis framework, thereby defining the scope of the survey (e.g., techniques for the specification of visual metamodels, connector types, and notation).

The initial framework has then been applied by two authors to perform an initial analysis of one specification. This resulted in a minor revision of the framework. This revision involved the analysis of the structure of specification documents, and the techniques for specifying constraints. The extended analysis framework was then applied by the authors to analyze the found specification documents. This analysis has been conducted individually, each author thereby created a list of identified techniques and survey results. In a final collaborative steps, the individual results have been condensed into a comprehensive set of specification techniques and results. Each technique has been additionally given an intuitive and expressive identifier.

Within the Google keyword search we were referred several times to pages such as Rosetta Standards[3] or Integrated DEFinition Methods (IDEF)[4] which enforce a registration before one can access the specifications. If the registration was free of charge we created an account in order to access the specifications. The Climate Science Modelling Language (CSML)[5] was not accessible and therefore it was 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.

All specifications meeting all inclusion criteria were classified as relevant. We read the heading and the introduction sections to ensure that the inclusion criteria IC 1 and IC 2 are fulfilled. We found two specification documents for Business Process Model and Notation (BPMN) and UML: one from the OMG and one from the International Organization for Standardization (ISO). In both cases, we opted for the former one. The Decision Modeling Notation (DMN) introduces two languages: one language - the Decision Requirements Diagram (DRD) - has a graph-based structure, while the second language - decision tables - has no such structure. We decided to consider the DMN in our survey - but only its DRD subset. In total, we found 17 relevant specifications using the keyword search of Google. On the websites of OMG and OpenGroup we identified *further* 6 relevant specifications which we also analyzed.

[3]https://resources.gs1us.org/rosettanet, last accessed: 2018-12-04
[4]http://www.idef.com/, last accessed: 2019-01-21
[5]http://csml.badc.rl.ac.uk/, last accessed: 2019-01-22

Table 1
Modeling language specifications surveyed in this paper

	Version	Maintaining Institution	Domain	Pages	Ref.
ArchiMate	3.0.1	Open Group	Enterprise Architecture	187	[78]
BPMN - Business Process Model and Notation	2.02	OMG	Business Processes	532	[58]
CMMN - Case Management Model and Notation	1.1	OMG	Case Management	144	[60]
DMN - Decision Model and Notation	1.2	OMG	Business Decisions	208	[63]
IFML - Interaction Flow Modeling Language	1.0	OMG	User Interactions	144	[59]
LML - Lifecycle Modeling Language	1.1	LML	Systems Engineering	70	[50]
OPM - Object Process Methodology	522	ISO	Automation Systems and Integration	183	[34]
S2ML - System Structure Modeling Language	1.0	OpenAltaRica	Prototypes	53	[64]
UML - Unified Modeling Language	2.5.1	OMG	Software Systems	796	[61]
URN - User Requirements Notation	3.0	ITU-T	Requirements Engineering	250	[35]
VDML - Value Delivery Modeling Language	1.1	OMG	Value Creation	137	[62]

In a second step, we evaluated the relevant specifications along the exclusion criteria by reading the introduction section and by cross-reading the following sections. Furthermore, we analyzed the table of contents to identify the scope of the specification. This evaluation step classified 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 visual modeling language. Also the Business Motivation Model (BMM) specification required further evaluation because of the limited notation[6]. The specifications belonging to this category were evaluated again and discussed by the authors to come to a final decision. Eventually, 11 specifications which comply with our search criteria have been identified - see table 1 for an overview.

4. Visual Metamodel Specification Techniques

All surveyed modeling language specifications except the System Structure Modeling Language (S2ML) introduce a metamodel that represents the syntax of the modeling language. However, there is a discrepancy with respect to the techniques used for specifying a metamodel. Most specifications use visual metamodels, whereas also matrix and tabular forms were found. S2ML only uses natural language and is therefore not in the scope of the following analysis.

Visual metamodel specification techniques specify the syntactic rules of a modeling language by graphical means. The following subsections introduce the identified techniques along generic descriptions on the one hand, and examples taken from the respective specification documents on the other.

4.1. Slicing Metamodels

Metamodels are often too large to be visualized and comprehended in a single figure [6]. Actually, not one of the surveyed specifications visualizes one complete visual metamodel. Specifications such as Unified Modeling Language (UML) [61] and ArchiMate [78] use a specification technique which will be referred to in the following as *slicing*. The complete metamodel is thereby separated into multiple slices. Each slice has at least one element which is part of another slice. An example is shown in figure 3, where the complete metamodel is decomposed into three slices. Using the redundant elements enables the re-construction of the complete metamodel by merging the slices.

A further distinction between *redundant slicing* and *non-redundant slicing* is employed. A non-redundant slicing example is shown in Figure 3 - *only one element of each slice is used in another slice*. In the redundant slicing approach, *several elements of one slice are part of at least one other slice*. All surveyed specifications use the redundant slicing approach. An example from the Business Process Model and Notation (BPMN) specification [58] is depicted in figure 4. In both slices, the attributes and the relationships between *BaseElement* and *Documentation* are introduced.

[6]http://www.omg.org/spec/BMM/1.3/PDF/, last accessed: 2019-01-21

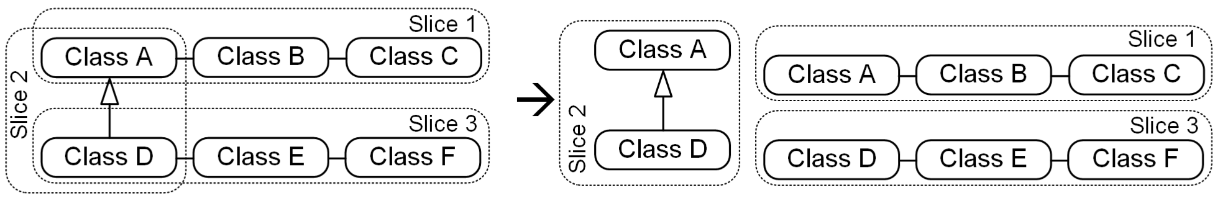
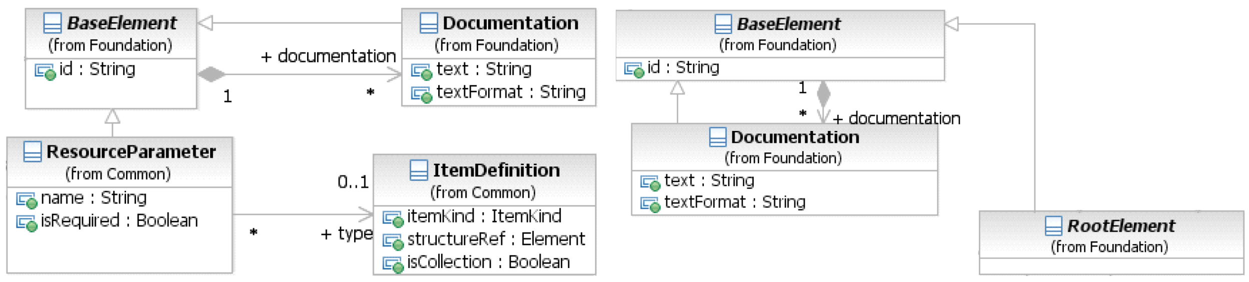


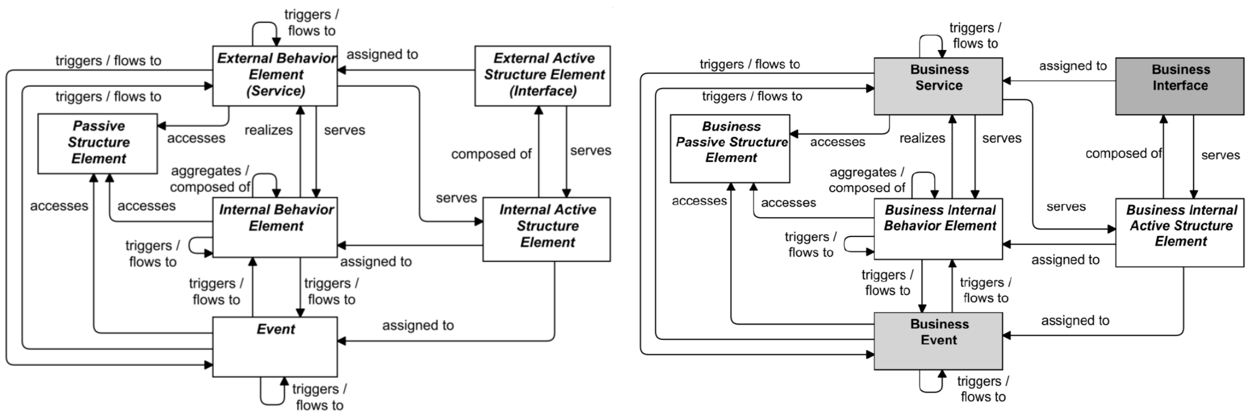
Figure 3: Metamodel decomposition into slices



(a) BPMN - slice A - adapted [58, p. 94]

(b) BPMN - slice B - adapted [58, p. 102]

Figure 4: BPMN slices introducing the metaelements *BaseElement* and *Documentation*


External Behavior Element (Service)
External Active Structure Element (Interface)
Passive Structure Element
Internal Behavior Element
Internal Active Structure Element
Event
Business Service
Business Interface
Business Passive Structure Element
Business Internal Behavior Element
Business Internal Active Structure Element
Business Event
triggers / flows to
assigned to
accesses
realizes
serves
composed of
aggregates / composed of

(a) ArchiMate generic metamodel [78, p. 13]

(b) ArchiMate business layer metamodel [78, p. 55]

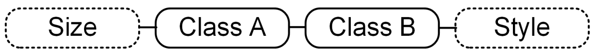
Figure 5: Generic metamodel and business layer metamodel of ArichMate [78]

4.2. Referencing Metamodels

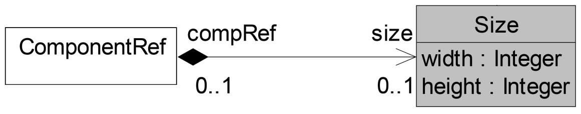
Slicing metamodels intrinsically introduce redundancy. In order to moderate this effect, several specifications such as the User Requirements Notation (URN) [35] use a special kind of slicing which will be referred to in the following as *referencing metamodels*. These referencing metamodels use *reference elements* that only contain the name of a referenced metamodel element in a different slice while omitting additional information such as attributes. One slice contains the complete specification of the metamodel element to which the reference elements only refer to.

4.3. Generic Metamodels

Generic metamodels do not describe syntactic elements of the modeling language - instead they introduce structure to the modeling language. ArchiMate [78] heavily uses generic metamodels. Figure 5a depicts the generic metamodel with the abstract metaelements while figure 5b shows the metamodel of the business layer which comprises concrete classes derived from the generic metamodel concepts in figure 5a and further generic metamodel concepts. For example, the concrete metaelement *Business Service* is derived from the generic metaelement *External Behavior Element (Service)*. Still, additional generic metaelements like *Business Passive Structure Element* are given.



(a) Metamodel with notation-aware metaelements



(b) URN metamodel - adapted [35, p. 60]

Figure 6: Generic structure and excerpt of a notation-aware URN metamodel [35]

Table 2
Generic example of the matrix specification technique; CT=connector type

	Class A	Class B	Class C	...
Class A	CT Z	-	-	...
Class B	-	CT Y	CT Y	...
Class C	-	CT Y	CT Y	...
...	...	...	...	...

Table 3
Generic example of the table specification technique

Class	**Parent Classes**	**Child Classes**
Class A	-	Class B
Attributes	**Data Type**	
Attribute A	String	
Connector Types	**Target**	
Connector Type A	Class C	
	Attribute	**Data Type**
	Attribute C	Integer
Connector Type B	Class D	

4.4. Notation-aware Metamodels

Usually, metamodels describe the (abstract) syntax of a modeling language without its notation (also referred to as concrete syntax). However, the survey revealed what will be defined as *notation-aware metamodels* which combine the specification of syntax with notation. Such metamodel specifications distinguish between conventional metaelements and specific metaelements for the notation. The latter metaelements have no semantics, they contain attributes that solely specify notational aspects [35, p. 6]. Figure 6a shows a generic structure of a notation-aware metamodel. *Class A* and *Class B* are conventional metaelements whereas the metaelements *Size* and *Style* are only specifying how objects of the connected metaelements shall be visualized in the model layer. For example, the style metaelement might have a boolean attribute which indicates if the corresponding objects are filled with a color.

The URN specification [35] heavily uses such notation-aware metamodels. Figure 6b shows an example where the metaelement *ComponentRef* has a connector to the notation-specific metaelement *Size* which specifies aspects of its visualization - the size.

4.5. Matrix Metamodels

Several of the analyzed specifications use matrices instead of or in addition to one of the previously introduced specification techniques. These matrices show on both axes the concrete object types of the modeling language as well as the allowed connector types which can connect them. A schematic overview of such a matrix is depicted in table 2.

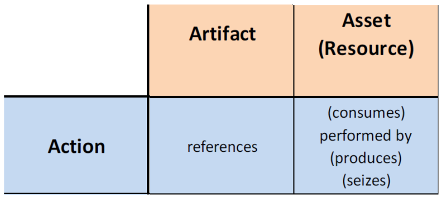
Figure 7a shows an excerpt of a matrix specification used in Lifecycle Modeling Language (LML) [50]. Action, Artifact, and Asset are object types. *Resource* is inheriting from *Asset* - indicated by the brackets. The connector types which are surrounded by the brackets - *consumes*, *produces* and *seizes* - are only valid for the sublcasses. Thus, *Resource* objects can be connected to *Action* objects by *consumes* connectors.

4.6. Tabular Metamodels

The tabular specification technique is related to the matrix specification in the sense that the metamodel is defined using a two-dimensional organization. A schematic example of the table specification is shown in table 3. All attributes and the connector types which can be used for an object type are described using a table. Such a tabular specification usually introduces redundancy. For example, connector types including their attributes which are used for connecting several object types have to be re-introduced for each class. Furthermore, readability and comprehension of tabular specifications are limited [55]. An example of the LML is depicted in figure 7b. Several other specifications use a simplified table where only the semantics of the object types is described. Thereby, the connector types and attributes are not defined within the table.

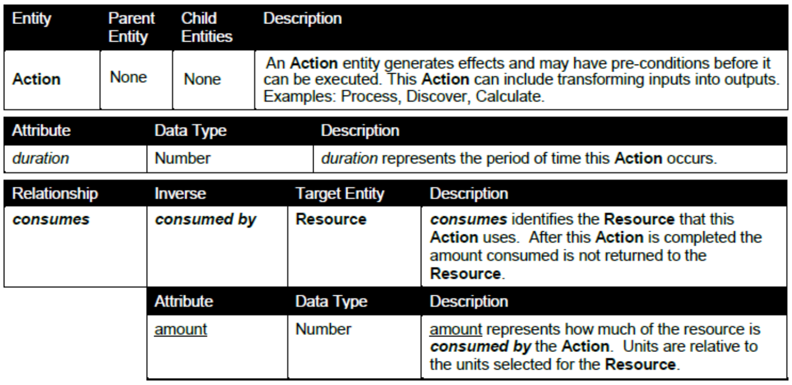
4.7. Usage of Metamodel Specification Techniques

Table 4 summarizes the usage of the previously introduced visual metamodel specification techniques throughout the survey. The slicing approach is widely used in the surveyed specifications. Only LML, S2ML, and Object Process



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

(a) Matrix specification used in the LML - adapted [50, p. 13]



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**.

(b) Table specification used in the LML - adapted [50, p. 15]

Figure 7: Table and matrix specification technique used in the LML

Table 4
Usage of visual metamodel specification techniques (⊕=used, -=not used); MM = Metamodel

	Slicing MM	Reference MM	Generic MM	Notation-aware MM	Matrix MM	Table MM	Used techniques
ArchiMate	⊕	-	⊕	-	⊕	⊕	4
BPMN	⊕	⊕	-	-	⊕	⊕	4
CMMN	⊕	⊕	-	-	-	⊕	3
DMN (DRD)	⊕	⊕	-	-	⊕	⊕	4
IFML	⊕	-	-	-	-	⊕	2
LML	-	-	-	-	⊕	⊕	2
OPM	-	-	-	-	-	⊕	1
S2ML	-	-	-	-	-	-	0
UML	⊕	⊕	-	-	-	⊕	3
URN	⊕	⊕	-	⊕	-	-	3
VDML	⊕	⊕	-	-	-	-	2
Used by	8/11	6/11	1/11	1/11	4/11	8/11	

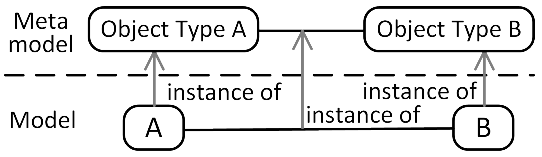
Methodology (OPM) - which only introduces a single visual metamodel overview in the specification - do not use slicing. Reference elements are used in six out of the eight specifications that use slicing. Notation-aware metamodels and generic metamodels are used by one specification, but in both cases, they are used heavily. Surprisingly, no specification uses a tree-based specification for the metamodels such as used for Eclipse-based metamodels [25].

As table 4 shows, specifications usually use a combination of several metamodel specification techniques whereby matrices and tables are often used to summarize the specifications - the only exceptions are the S2ML, URN, and Value Delivery Modeling Language (VDML). On average, 2.54 techniques are used, the median of applied specification techniques in one specification document is three.

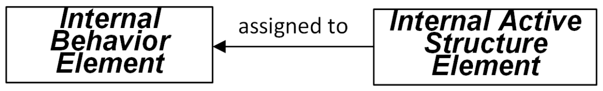
4.8. Analysis of Visual Metamodel Specification Techniques

The previous subsections comprehensively introduced and showcased the identified visual metamodel specification techniques. The following provides a qualitative evaluation that aims to describe the potential strengths and weaknesses of each technique, ultimately guiding the selection of the most appropriate technique in a given situation.

Slicing Metamodels The strengths of this technique are realized by the principles of separation of concern and divide & conquer. The former refers to the possibility of separating a large overarching metamodel into separate slices which might cover certain aspects. The latter refers to handling complexity when designing and comprehending a large metamodel. By designing and comprehending separate slices, ultimately the whole metamodel is designed/comprehended more efficiently [6]. A drawback of this approach is that slicing naturally introduces redundancy which needs to be handled.



(a) Connector type specification by a connector in the metamodel (generic)



(b) Specification of *assigned to* connector type in ArchiMate - adapted [78, p. 13]

Figure 8: Sample specifications of a connector type by a connector

Reference Metamodels This technique shares the strengths of the slicing technique while mitigating the redundancy drawback. Moderating as there is still redundancy in the slices, however, only changes in the name of metaelements need to be kept consistent.

Generic Metamodels This technique is very useful for specifying a very large metamodel. In such situations, it is meaningful to provide a structure of the metamodel by means of generic elements before specifying all metamodel concepts in detail. A drawback is the mixing of regular modeling language concepts and concepts of the generic metamodel.

Notation-aware Metamodels The strength of this approach is related to the single source of information principle as the metamodel already comprises the notation (i.e., the concrete syntax) of the metaelements. Moreover, including the notation in the metamodel might also contribute to a more efficient comprehension of the modeling language. A drawback is that metamodels are overcoded by notational aspects, impeding the comprehension of the modeling language's syntactic nature.

Matrix Metamodels The strength of the matrix specification type is on providing a comprehensive specification of the allowed connector types between a pair of object types (see Section 5.5). The matrix specification technique also has several drawbacks. For example, the representation of class hierarchies is limited. Further, information such as relationship multiplicities and multiple endpoints cannot be represented.

Table Metamodels This technique is powerful for specifying the semantics of metaelements by means of attributes. However, tables are not well-suited for visualizing hierarchies of metaelements or compositions.

5. Connector Type Specification Techniques

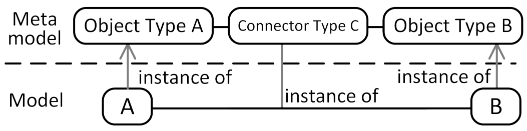
The term connector refers on the model layer to the construct which connects two or more object type instances. The specification of a connector type in the metamodel layer might seem to be trivial - i.e., by using a line between the two object types - but the survey revealed different specification techniques. In total, four major techniques with three sub-techniques for the specification of a connector type in a visual metamodel have been identified which will be discussed in the following.

5.1. Connector Type Specification by a Connector

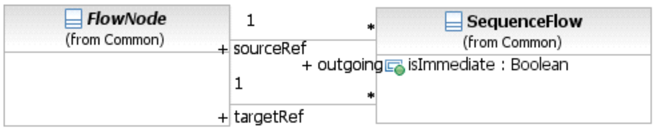
In this approach, a connector type is specified by a connector which relates two object types in the metamodel. A connector type specified like this usually has no attributes. An example is depicted in figure 8a where the distinction between connector types and object types is visually encoded. Figure 8b shows an excerpt of the ArchiMate metamodel where the connector tpye *assigned to* is specified by a connector. Instances of it connect instances of the object type *Internal Active Structure Element* with instances of the object type *Internal Behavior Element*. The object types represent the start- and the endpoint of instances of the connector type.

5.2. Connector Type Specification by a Connector Type

By following this approach, a connector type is specified as a connector type which might have attributes providing references to object types. Such an attribute is referred to in the following as *reference attribute*. In the generic example visualized in figure 9a, *Connector Type C* represents a connector type while A and B are object types. The lines



(a) Connector type specification by a connector type



(b) *SequenceFlow* connector type specified by a connector type - adapted [58, p. 144]

Figure 9: Sample specifications of a connector type by a connector type

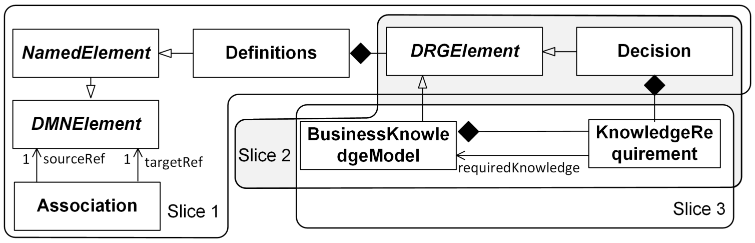


Figure 10: Excerpt of the DMN metamodel [63]

connecting *A* with *C* and *C* with *B* are reference attributes which act as the start and end of a connector type. This differs from the lines in the metamodel depicted in figure 8a where the lines represent connector types.

To foster understanding of this technique, an example from the Business Process Model and Notation (BPMN) specification [58] is depicted in figure 9b, showing two metaelements which are connected with two reference attributes. However, figure 9b does not reveal that *SequenceFlow* is a connector type and *FlowNode* is an object type. Only an additional natural language description specifies this important detail.

5.3. Variations of Connector Type Specification by a Connector Type

Further identified variants for the specification of connector types are explained using the Decision Modeling Notation (DMN) specification [63] - see figure 10. Here, the metaelement *Association* represents a connector type which connects instances of the object type *DMNElement*. Again, it is described in natural language that the metaelement *Association* represents a connector type. Similar to the *SequenceFlow* element illustrated in figure 9b, it has two reference attributes: source and target. Also, the metaelement *KnowledgeRequirement* of the metamodel represents a connector type, but uses a different structure. This is because the metaelement *Knowledge Requirement* has only a single reference attribute *requiredKnowledge* which represents the type of the endpoint.

It needs to be clarified that *BusinessKnowledgeModel* is a typical object type - not a model. The starting point of the connector type is the object type *BusinessKnowledgeModel* that composes *KnowledgeRequirement*. This is different from metaelements such as *Association* discussed before, where no composition is used. In the case of *Association*, the instances of the connector type are independent - in the case of the *KnowledgeRequirement* existence of its instances is bound to the instances of the type which is defined as startpoint. The DMN specification [63] uses such an approach also for the specification of other connector types. Consequently, the DMN specification uses specification variants with and without composition for defining connector types. The variant without composition is referred to as *Connector type specified by a connector type without composition*, while the variant with composition is referred to as *Connector type specification by a connector type with composition* in the following.

A third variant of this approach uses a special label for a metaelement that represents a connector type, referred to in the following as *connector type label*. An excerpt of the Object Process Methodology (OPM) metamodel specification is depicted in figure 11. Here the connector type representation is different from usual metaelements by the usage of the suffix *Link* in its label. As figure 11 shows, the connectors with the label *connects* indicate which object types are start- and endpoint of a connector type.

5.4. Connector Type Specification in Natural Language

This technique completely omits connector types in the visual metamodel. Instead, a natural language description introduces connector types. Examples are depicted in the slicing metamodel created based on the Case Management

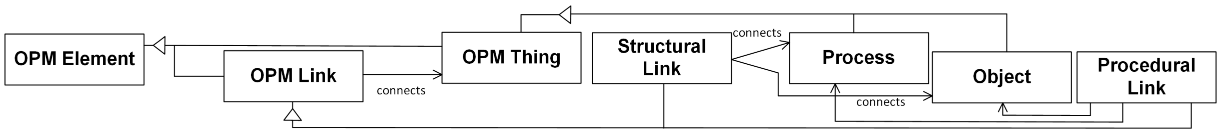
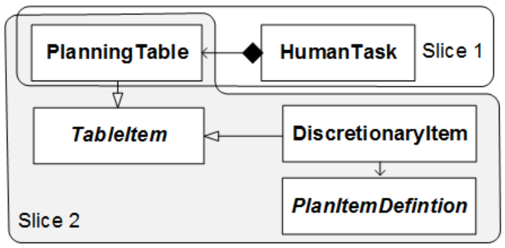
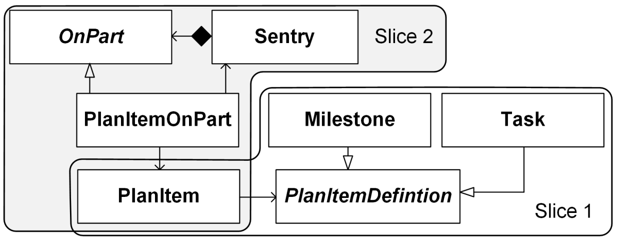


Figure 11: Connector types in OPM - adapted from [23, p. 380], referring to [34, p. 13]

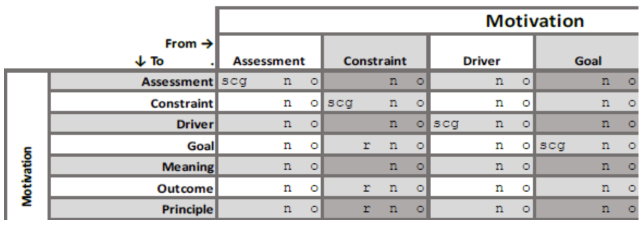


(a) CMMN metamodel - Example 1



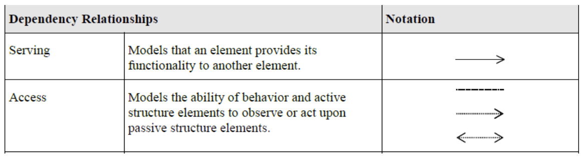
(b) CMMN metamodel - Example 2

Figure 12: CMMN metamodel [60] slices where connector types are not specified



		Motivation			
	From → ↓ To	Assessment	Constraint	Driver	Goal
Motivation	Assessment	scg n o	n o	n o	n o
	Constraint	n o	scg n o	n o	n o
	Driver	n o	n o	scg n o	n o
	Goal	n o	r n o	n o	scg n o
	Meaning	n o	n o	n o	n o
	Outcome	n o	r n o	n o	n o
	Principle	n o	r n o	n o	n o

(a) Matrix connector type specification in ArchiMate [78, p. 122]



Dependency Relationships		Notation
Serving	Models that an element provides its functionality to another element.	⟶
Access	Models the ability of behavior and active structure elements to observe or act upon passive structure elements.	-------- ⟶ ⟷

(b) Table connector type specification in ArchiMate [78, p. 35]

Figure 13: Matrix and table specification technique used in ArchiMate

Model and Notation (CMMN) specification [60] visualized in figure 12. All lines between metaelements - e.g., between *DiscretionaryItem* and *PlanItemDefinition*, represent reference attributes. The only exceptions from this rule are the inheritance connectors, e.g., between *TableItem* and *PlanningTable*. CMMN specifies a connector type between the object types *HumanTask* and *DiscretionaryItem*. However, in the visual metamodel depicted in figure 12a, no such connector type is specified. There is only an indirect relationship via the metaelement *PlanningTable*.

Similarly, figure 12b shows another example where a connector between *Sentry* and *OnPart* is visualized. *OnPart* is an abstract object type. With this, the specification defines a connection on the model layer between Sentries and PlanItemDefinitions. Again this connector type is not defined in the visual metamodel.

5.5. Connector Type specification by a Table or Matrix

The table and matrix specification types were already introduced in section 4. For the specification of connector types, matrices are widely used as they enable the compact visualization of all allowed connector types between the given object types. Similarly as for object types, table specifications are used to specify the semantics and/or the notation of a connector type. Figure 13 shows the usage of both specification techniques in ArchiMate. Figure 13a exemplifies how ArchiMate uses the cells that intersect two object types to list abbreviations (e.g., (a)ccess, (c)omposition, a(g)gregation, i(n)fluence, ass(o)ciation) of all connector types that are allowed to connect objects of these object types. Figure 13b shows the specification of the semantics and graphical representation of connector types.

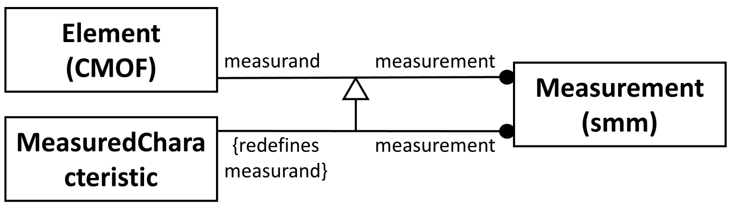
5.6. Usage of Connector Types Specification Techniques

The survey showed, that also for the specification of connector types heterogeneous techniques are used in practice. Moreover, most surveyed specifications use a combination of techniques. Table 5 summarizes the used connector type specification techniques surveyed in visual metamodels.

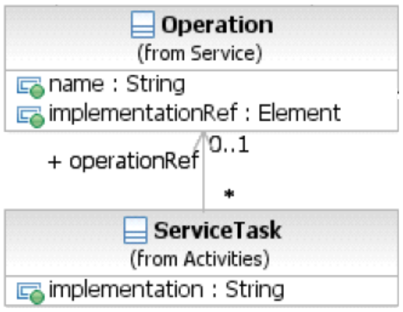
Table 5
Connector Type specification techniques (⊕=used, -=not used), CT=Connector Type

	CT spec. by a Connector	CT specification by a			CT spec. in Natural Language	CT spec. by a Table or Matrix	Used techniques
		CT without Composition	CT with Composition	CT label			
ArchiMate	⊕	⊕	-	-	⊕	⊕	4
BPMN	-	⊕	-	-	⊕	⊕	3
CMMN	-	⊕	-	-	⊕	⊕	3
DMN (DRD)	-	⊕	⊕	-	⊕	⊕	4
IFML	-	⊕	-	-	-	⊕	2
LML[1]	⊕	-	-	-	-	⊕	2
OPM	-	-	-	⊕	⊕	⊕	3
S2ML[2]	-	-	-	-	⊕	-	1
UML	⊕	⊕	⊕	-	⊕	⊕	5
URN	-	⊕	-	-	⊕	-	2
VDML	⊕	⊕	⊕	-	⊕	⊕	5
Used by	3/11	8/11	3/11	1/11	8/11	9/11	

[1] uses Entity-Relationship-Diagrams to specify some connector types
[2] selected relationships are specified in natural language



(a) Inheritance of a connector type in VDML - adapted [62, p. 55]



(b) Object type with reference attribute pointing to another object type - adapted [58, p. 93]

Figure 14: Specification examples of VDML and BPMN

Table 5 shows that all specifications except the System Structure Modeling Language (S2ML) use two or more connector type specification techniques. Value Delivery Modeling Language (VDML) [62] and Unified Modeling Language (UML) [61] use five different techniques while ArchiMate [78] and DMN [63] use four. On average, 3.09 specification techniques are used within one modeling language specification (median of 3).

5.7. Analysis of Connector Type Specification Techniques

The surveyed specification techniques all come with strengths and weaknesses. In the following, an analysis is presented that yields toward guidelines for the selection of the most appropriate specification technique.

by a connector: This technique simplifies the comprehension of metamodels as there is a strict distinction between object types and connector types. The drawback of this approach is its limited expressiveness and flexibility. For example, the visualization of connector type hierarchies is limited as well as the visualization of connector type attributes. Indeed the survey found such a connector type hierarchy in the VDML specification [62] which is depicted in figure 14a. The VDML connector type connecting *Element (CMOF)* and *Measurement (smm)* inherits its properties to the connector type connecting *MeasuredCharacteritic* with *Measurement (smm)*.

by a Connector Type: The benefit of such metamodels is that attributes and inheritance relationships can be easily specified for connector types as they are treated as usual object types in the metamodel. However, specifications often do not reveal which metaelements represent a connector type and which ones do not. Additional specifications, e.g. using natural language, need to specify this very important information. Otherwise, visually distinguishing between the two types of metaelements is not possible.

by a Connector Type Label: The usage of separate connector type metaelements has the advantage that no additional textual description is necessary defining that a metaelement represents a connector type. As a drawback, these specific metaelements need to be introduced as they might share the same notation as object types and therefore hamper intuitive visual differentiation.

by Natural Language: The benefit of the textual description is that arbitrary connector types can be defined without overloading the visual metamodel. Thus, contributing to an ease of comprehension of the metaelements. However, inheritance as well as attributes of connector types are difficult to be specified in natural language. Moreover, each aspect specified in textual language needs to be kept consistent with the visual metamodel.

by a Table: Table specifications enable the compact specification of the attributes of a connector type. Moreover, tabular specifications are often used to map the connector type to its semantics and to its notation.

by a Matrix: Matrix specifications provide a compact visualization of the allowed connector types between a pair of object types. As a drawback, such matrices only enable binary relationships and do not allow inheritance between connector types.

Eventually, also one anti-pattern shall be reported. One major issue surveyed was the representation of references in visual metamodels by means of connectors between metaelements. Such references can be considered as an association between metaelements. In contrast to connector types which become connectors in the model layer, references are attributes of one object type that enable the definition of an association to another object type. This inevitably hinders intuitive understanding of the specification by means of differentiating between connector types and reference attributes.

An example is depicted in figure 14b referring to the BPMN specification [58]. It shows the object type *ServiceTask* which has a reference attribute to the object type *Operation*. However, this information is not formally specified in the metamodel. One needs to see instantiations - as in figure 9a - or additional textual specifications needs to be provided - as for BPMN.

6. Metamodel Concepts

Elements of a metamodel are instances of elements of the corresponding meta-metamodel [75], e.g., a metamodel connector type is inherited from a meta-metamodel concept *Relationship Type*. However, while e.g. the Unified Modeling Language (UML) specification describes that it is based on the MetaObject Facility (MOF) meta-metamodel, other specifications such as ArchiMate do not mention a meta-metamodel. The literature only covers surveys of concepts of metamodeling platforms such as Eclipse EMF, MetaEdit+ [43], or ADOxx [7] [8]. Following research question 1.2, this section surveys the concepts which are specified in visual metamodels of modeling language specifications.

6.1. Analysis Criteria

The survey is based on concepts introduced in [47, 43]. A brief summary of the used concepts and their mapping to the scope of this analysis is given in the following:

First Class Concepts. (i) *Object Type*[8]: classes of objects which share the same connectors and attributes. (ii) *Connector Type*[8]: class of connectors which have the same attributes and use the same object types as endpoints. (iii) *Abstract Type*[9]: A metaelement that cannot be instantiated is coined abstract type. (iv) *Attribute*[8]: Metaelements have attributes representing their properties. (v) *Role*[8]: A role clarifies the position of object types which are connected to other object types with a connector.

Relationships (between object types). No distinction between connector types and reference attributes is applied in the following. (i) *Arity*[10]: Describes the number of endpoints of a connector in the metamodel. For example, binary connectors have two endpoints, ternary connectors have three endpoints etc.. (ii) *Multiplicity*[8]: Defines the minimal and maximal number of relations between objects. (iii) *Direction*[11]: Connectors between object types can

[7]ADOxx metamodeling platform [online]: `www.adoxx.org`, last accessed: 2019-05-14

[8]concept from [47] and [43]

[9]concept added by the authors

[10]concept from [43]

[11]concept from [47]

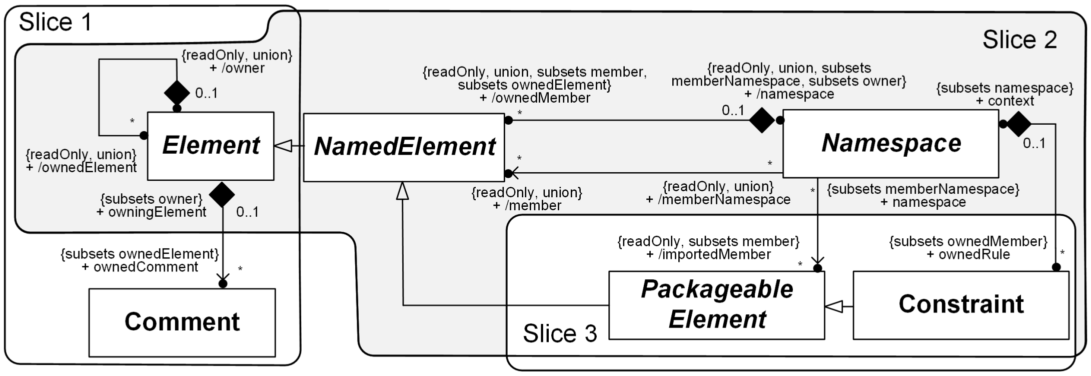

Slice 1
Slice 2
Slice 3
Element
NamedElement
Namespace
Comment
Packageable Element
Constraint
{readOnly, union} + /owner
{readOnly, union} + /ownedElement
{subsets owner} + owningElement
{subsets ownedElement} + ownedComment
{readOnly, union, subsets member, subsets ownedElement} + /ownedMember
{readOnly, union, subsets memberNamespace, subsets owner} + /namespace
{subsets namespace} + context
{readOnly, union} + /member
{readOnly, union} + /memberNamespace
{subsets memberNamespace} + namespace
{readOnly, subsets member} + /importedMember
{subsets ownedMember} + ownedRule
0..1
*

Figure 15: Excerpt of UML metamodel [61] using the union and subset concepts (see brackets '{', '}')

be directed from a source to a target object type. (iv) *Inverse*[9]: Inverse connectors are created automatically by interchanging source and target of a referred connector. (v) *Composition*[10]: Composition connectors represent strong whole-part relationships similar to the composition in UML class diagrams. The instance of the metaelement which represents the *whole* is called owner while the instance of the metaelement which represents the *part* represents the child. (vi) *Recursive Connectors*[9]: For recursive connectors, source and target refer to the same metaelement. (vii) *Redefines/Subset/Union*[11]: These attributes are usually added to connectors in the metamodel as shown in figure 15. (a) *Redefines*. Connectors that refer to another connector which is hidden by the connector with the redefinition flag. (b) *Subset*. Two objects which are related using a subset connector are automatically additionally related with the connector to which the subset connector refers. (c) *Union*. The inverse of the subset connector is the union connector. (viii) *Ordered*[11]: An ordered attribute is usually added to connectors in the metamodel similar to the redefines, subset or union attributes which leads to an ordered collection in the model. (ix) *Derived Reference Attribute*[9]: This attribute is only available for metamodels where the connectors in the metamodel represent reference attributes. Thereby, the reference attribute is derived e.g. by other attributes which is indicated by a backslash as exemplified in figure 15.

Attribute. (i) *Multiplicity*[8]: Attributes which represent a collection of elements. Multiplicity defines the minimal and maximal number of elements of this collection. (ii) *Unique*[8]: Unique attributes need to have a unique value in the model. (iii) *Default Value*[10]: Attributes optionally have a default value. (iv) *Derived Attribute*: Attributes which are derived e.g., from other attributes.

Inheritance[10]. Metamodel elements might be related to other metamodel elements using an inheritance relationship in order to establish a hierarchy. Similar to software engineering, *single* and *multiple inheritance* can be distinguished for single and multiple super classes, respectively.

Others[9]. Besides the metaelements, *enumeration classes* can be part of metamodels which act as specification of a datatype for an attribute.

The concepts *acyclic* and *unshared*, introduced in [47], are not listed, as they were not used in any of the surveyed specifications. Similarly, several concepts derived from the meta-metamodel concepts introduced in [43] were not identified: port type, model type, links to model types, grouping, identification, role type, dependency and constraint language.

6.2. Analysis Results

The results of analyzing the modeling language specifications according to the previously introduced criteria are summarized in table 6. Due to the absence of visual metamodels in System Structure Modeling Language (S2ML) and Lifecycle Modeling Language (LML), the results of these two specifications are not comparable with the others. The results show that the usage of metamodel concepts in visual metamodels of specifications is heterogeneous. Only two concepts are used in all metamodels: *object types* and *binary connectors used for connecting object types*. However, the semantics of these connectors is different - they represent either connector types or reference attributes as discussed in section 5.

The ArchiMate specification [78] uses no attributes in its visual metamodel. The connectors between object types are used for defining connector types. Due to the absence of attributes, ArchiMate does not use the metamodel concept enumeration. A profiling mechanism exists which can be used for adding attributes to metaelements. Only one

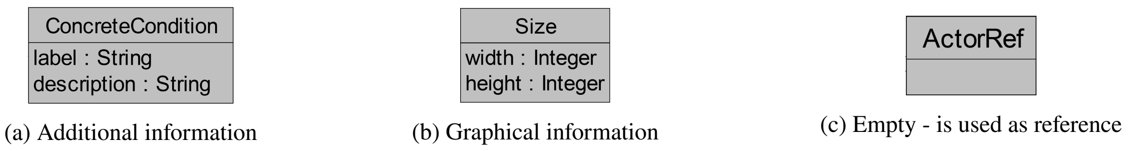


Figure 16: Metaelements used for notation specification in the User Requirements Notation (URN) [35]

composition connector is used in the metamodel. Also, multiplicities are used rarely.

The Business Process Model and Notation (BPMN) specification [58] uses unique attributes as well as default values and ordered collections for attributes. However, these concepts are not used in the visual metamodel of BPMN. They are defined in natural language and in tables which describe the visual metamodel. The BPMN specification uses derived reference attributes in visual metamodels, their names are prefixed with a slash.

The Case Management Model and Notation (CMMN) specification [60] uses attribute tables in which unique attributes are defined. The concept is used in the attribute tables but not in the visual metamodel. Also, derived attributes are not used in CMMN.

The Decision Modeling Notation (DMN) specification [63] uses derived reference attributes in visual metamodels which are prefixed with a slash. Additionally, default values are used for the attributes.

The Interaction Flow Modeling Language (IFML) specification [59] introduces enumerations in the specification, however they are not introduced in the visual metamodel. Further, the specification describes that some attributes are unique - which are not declared as unique in the visual metamodels.

In the LML specification [50] a visual metamodel does not exist, limiting the comparison with the other specifications. LML is the only surveyed specification that uses inverse connector types.

The visual metamodel of Object Process Methodology (OPM) [34] is declared as an overview. While it does not use attributes or default values, they are defined in natural language. Moreover, the OPM specifies unique attributes, e.g., each element should have a unique name, which is not defined in the visual metamodel. Inverse relationships are introduced using a table.

The S2ML specification [64] neither provides visual metamodels nor tables and matrices. Hence, an evaluation according to the given metamodel concepts was not possible.

The UML specification [61] uses union, subset and redefines concepts as the created slicing metamodel excerpt depicted in figure 15 shows. This example shows that the UML extensively uses constraints to specify syntactic rules on top of the visual metamodel. Between *NamedElement* and *Namespace* is a connector with the roles *member* and *memberNamespace*. Both of them are derived. The second connector between these two object types uses the roles *ownedMember* and *namespace*. Both of them are subsets of the roles of the previously described connector as well as of the recursive connector of the metaelement *Element* with the roles *owner* and *ownedElement*. At the same time they are derived, too, which implies that there exists another connector which subsets the roles *ownedMember* and *namespace*. In addition, the UML is the only specification that uses the concept *readOnly*, indicating a write protection to connectors.

The URN specification extends conventional metaelements with specific metaelements that do not directly specify the syntactic nature of the language. Default values as well as ordered reference attributes are used in visual metamodels of the URN. The ordered collections are marked with the keyword *ordered* surrounded by curly brackets and are used for reference attributes.

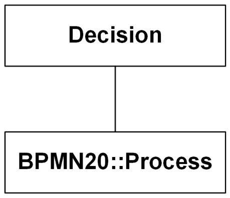
Three different types of these special metaelements have been classified (see figure 16). (i) Containing attributes with additional information. The *description* attribute of the class depicted in figure 16a is not used for the notation while the others are. (ii) Containing attributes used for the notation, e.g., the size attributes depicted in figure 16b. (iii) Not containing any attributes. It is used to visualize the references of the metaelement. For example, the object type *Actor* has multiple references to the object type *ActorRef* which allow different visualizations of the same actor.

The Value Delivery Modeling Language (VDML) specification [62] uses ordered reference attributes and the redefines concept in conjunction with connectors in its visual metamodel. Subset and union concepts are not used. Additionally, only the VDML specification introduces metaelements for the purpose of executing simulations. For example, *a CalendarService can be used to determine resource availability with more precision. This is relevant for simulation. Specification of calendar structure is not in scope of the VDML* [62, p. 30]. Interestingly, the simulation algorithm itself is not introduced in the specification. Additonally, the VDML uses the redefines concept.

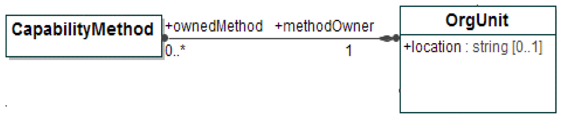
Table 6: Concepts used in visual metamodels ((y)es = supported, (n)o = no support)

	ArchiMate	**BPMN**	**CMMN**	**DMN (DRD)**	**IFML**	**LML**[1]	**OPM**	**S2ML**[3]	**UML**	**URN**	**VDML**
First Class Concepts											
Object Types	y	y	y	y	y	y	y	n	y	y	y
Connector Types	y	n	n	n	n	y	y	n	n	n	y
Abstract Types	y	y	y	y	y	n	n	n	y	n	y
Attribute	n	y	y	y	y	y	n	n	y	y	y
Role	y	y	y	y	y	y	n	n	y	y	y
Relationship (between Object Types)											
Arity	binary	binary	binary	binary	binary	binary	binary	n	binary	binary	binary
Multiplicity	y	y	y	y	y	n	n	n	y	y	y
Direction	y	y	y	y	y	y	y	n	y	y	y
Inverse	n	n	n	n	n	y	n	n	n	n	n
Composition / Aggregation	y[2]	y	y	y	y	n	n	n	y	y	y
Recursive Connectors	y	y	y	n	y	y	n	n	y	y	y
Subset / Redefines / Union	n	n	n	n	n	n	n	n	y	n	y
Ordered	n	n	n	n	n	n	n	n	y	y	y
Derived Reference Attribute[4]	n	y	n	y	n	n	n	n	y	n	n
Attribute											
Multiplicity	n	n	y	y	y	n	n	n	y	n	y
Unique	n	n	n	n	n	n	n	n	n	n	n
Default Value	n	n	y	y	y	n	n	n	y	y	y
Derived Attribute	n	n	n	n	n	n	n	n	y	n	n
Inheritance											
Single/Multiple Inheritance	single	multiple	single	single	multiple	single	single	n	multiple	single	single
Others											
Enumerations	n	y	y	y	n	n	n	n	y	y	n

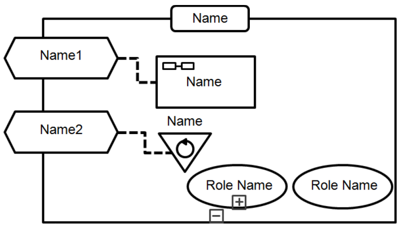
[1] no visual metamodel available - table introduced in the specification was used for evaluation
[2] one occurrence
[3] no metamodel available
[4] only available in metamodels where connectors represent reference attributes



(a) Connector to BPMN object type in the DMN [63, p. 51]



(b) Excerpt of the VDML metamodel which describes composition element *OrgUnit* - adapted [62, p. 46]



(c) VDML model which uses an OrgUnit - adapted [62, p. 92]

Figure 17: Cross-metamodel references and representations of composition in VDML

6.3. Further Observations and Analysis

Additionally to the previous discussion, the survey revealed further insights with regards to specfied aspects and used techniques which will be discussed in the following.

6.3.1. References to External Metaelements

Some specifications use metaelements which are introduced in other specifications. Figure 17a shows a connection between the DMN object type *Decision* to the BPMN object type *Process*. It seems like such inter-specification links are introduced by using *prefixes*. The VDML and BPMN specifications also use this technique as shown in figure 14a. Interestingly, the specifications do not discuss these references to external metaelements.

6.3.2. Composition

All surveyed modeling languages use composition objects on model level. BPMN, DMN, IFML, UML, VDML, URN and CMMN use natural language to specify that metaelements act as composition types - there is no dedicated metaelement for specifying a composition type. An example of a visual specification of composition in the metamodel is depicted in figure 17b where the composition element *OrgUnit* is specified in the metamodel. Figure 17c then shows a sample model comprising an instance of *OrgUnit* containing other elements.

6.3.3. Mixing Metamodel and Model Layer

The survey also revealed cases in which concepts belonging to the meta-metamodel are used in the metamodel. ArchiMate specifies and uses (i.e., instantiates) connector types in a single metamodel as figure 18 illustrates. In slice 3, two connector types are defined: *assignment* and *aggregation* which are used in slice 4 to connect the metaelements *Collaboration* and *Internal Behavior Element* with the metaelement *Internal Active Structure Element*. Hence, the metamodel hierarchy which clearly separates specification from instantiation to different meta levels [75] is blurred.

Further, the specification that a connector type (i.e., a named relationship in ArchiMate) has a source and a target - see the top of figure 18 - is usually part of the meta-metamodel. In addition to ArchiMate, only the OPM specification blurres the meta levels in this way. The OPM metamodel (see figure 11) introduces the metaelements *OPM Link* and *OPM Thing* which are connected using a connect connector which is usually defined in the meta-metamodel. The reason for this might be the absence of a meta-metamodel in those languages: While languages such as UML are using MOF[12] as meta-metamodel, OPM and ArchiMate do not refer to a meta-metamodel explicitly.

The usage of abstract metaelements such as *Relationship* enable the specification of a hierarchy of connector types. For example, in figure 18 the connector types *Assignment* and *Aggregation* are both sub-classes of the connector type *Structural Relationship* which itself is a subclass of *Relationship*.

6.3.4. Reference Attribute Ownership

VDML, UML and IFML use a dot symbol at the connector ends (metaelements) in the visual metamodel to indicate if a reference attribute - which is introduced with the connector - belongs to this metaelement. A connector with dots at both ends indicates that both metaelements have a reference attribute to the connected metaelement (see the connector between *Namespace* and *Constraint* in figure 15). This dot notation is also used in UML class diagrams [61, p. 202] to distinguish between class-owned and association-owned ends.

[12]http://www.omg.org/mof/, last accessed: 2019-02-04

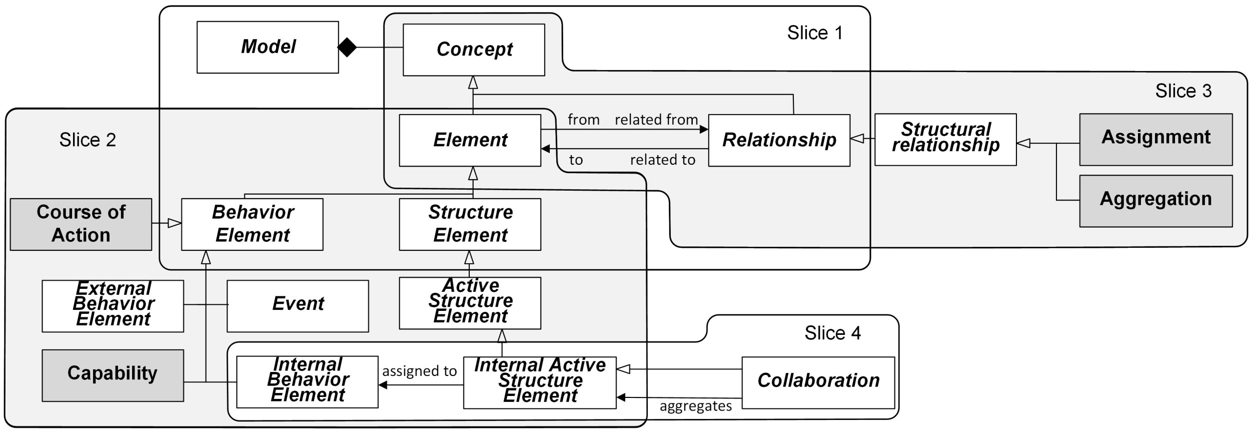


Figure 18: Specification (slice 3) and usage (slice 4) of connector types in the same ArchiMate metamodel [78]

6.3.5. Constraint Specification Techniques

All surveyed specifications use constraints to restrict the valid relationships between object types and connector types. For example, the BPMN specification [58, p. 174] describes that *an Event Sub-Process MUST have one and only one Start Event*. This constraint is not expressed in the visual metamodel of BPMN. This is also true for the following exemplary constraint of the VDML specification [62, p. 32]: *The Role that performs an Activity MUST be contained in the Collaboration that also contains the Activity*. The URN specification [35] shows that constraints can also be used for attributes of metaelements. URN specifies that the attribute *threshold shall evaluate to a non-negative Integer value, or it may be empty, in which case it is deemed to evaluate to 0* [35, p. 71].

The survey revealed that a lot of constraints are specified in the visual metamodel. An example of the DMN specification is depicted in figure 20a. The example as well as a natural language constraint indicate that the metaelement *AuthorityRequirement* is a binary connector type which connects for example an instance of the object type *Decision* with an instance of the object type *KnowledgeSource*. However, the given metamodel in figure 20a indicates that this connector can have more than two targets. The binary characteristic of this connector type could be expressed in the visual metamodel by introducing abstract superclasses.

UML [61] and IFML [59] are the only specifications surveyed which additionally use non-natural language constraints. They rely on the Object Constraint Language (OCL). A complete analysis of constraints as well as the identification of optional constraints is out of scope of this paper and part of our further research.

7. Notation Specification and Evaluation

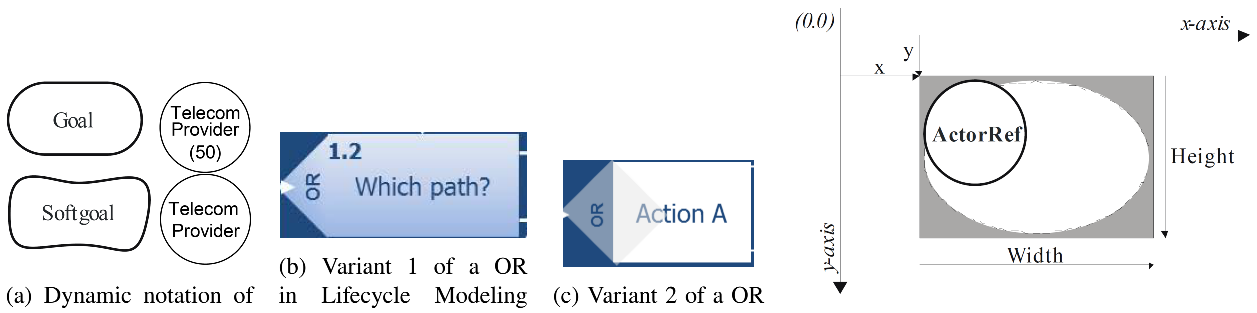
The notation of a modeling language, i.e., its graphical visualization, plays an important role for the purpose of communication and understanding [56]. This section provides a Survey of the found techniques regarding the specification of the notation (in section 7.1). Section 7.2 then provides an analysis of these techniques and an evaluation of the visual expressiveness of the surveyed modeling languages. Eventually, section 7.3 discusses some further observations.

7.1. Notation Specification Techniques

The analysis revealed three notation specification techniques. Furthermore, a distinction between dynamic and static notation as introduced by [11] is employed. *"If the notation of a languages element is fixed at all times, we refer to a static notation, if the notation can change depending on the current state (i.e. attribute value) of the element, we refer to a dynamic notation"* [11, p. 3402]. Lastly, alternative notations [31] and the fact that a lot of the surveyed modeling languages specify at least one notation conformance level are recognized. In the following, all six notation-related specification aspects are introduced briefly [13]:

Notation Samples Sample models are used to introduce the notation of modeling constructs.

Natural Language Notation Guidelines Natural language is used to introduce the notation of modeling constructs, i.e. by stating that *"metaelement x is represented by a blue rectangle"*.



(a) Dynamic notation of the actors and intentions - adapted [35, p. 21-23]

(b) Variant 1 of a OR in Lifecycle Modeling Language (LML) - adapted [50, p. 52]

(c) Variant 2 of a OR in LML - adapted [50, p. 51]

(d) Notation specification in **URN** - adapted [35, p. 49]

Figure 19: Different notation specification examples

Coordinate System A coordinate system is used to precisely specify height and width of modeling constructs, and optionally the positioning of labels. Figure 19d shows the specification of an Actor in URN.

Dynamic Notation A notation that reflects current attribute values. Figure 19a shows an example of a dynamic notation. The left two boxes represent instances of the object type *Intentional* which have two different values for the attribute *type*. Similar, on the right side, two instances of the object type *actor* are depicted. Here the value of the attribute importance is only visualized, if it is higher than 0.

Alternative Notation Optional and/or alternative notations are used to enable the user-specific or context-specific customization of the visual representation. Alternative notations are introduced in almost all of the surveyed specifications. ArchiMate heavily uses alternative notations for almost all modeling constructs. The modeler can decide among an iconic notation and rectangle that has the iconic element visualized in the upper corner of the rectangle. Figures 19b and 19c show alternative notations of the modeling construct *Or* in the LML.

Conformance Level Used to enforce adherence to the mandatory notation guidelines. Conformance levels are particular important for tool-vendors aiming to realize software support for a modeling language. Different levels of conformity can be distinguished e.g., based on the set of specification aspects covered - see e.g. [78, p. 1f.].

7.2. Usage of Notation Specification Techniques & Visual Expressiveness Evaluation

7.2.1. Analysis of Specification Technique Usage

Table 7 summarizes the key findings. It can be derived, that the used techniques for specifying the notation of a modeling language are very homogeneous. Interaction Flow Modeling Language (IFML) and LML use only the notation samples. All other specifications use sample visualizations supplemented with natural language notation guidelines. For example the Business Process Model and Notation (BPMN) specifies the notation of the object type *Task* as: *A task is a rounded corner rectangle that must be drawn with a single thin line* [58, p. 154]. Only the URN specifies the notation using a coordinate system (see figure 19d).

A more heterogeneous picture establishes when looking at the additional aspects covered in the notation specification. Dynamic notations are specified for six surveyed languages. Nine out of the 11 surveyed specifications introduce at least one alternative notation. Conformance levels are specified for eight modeling language notations.

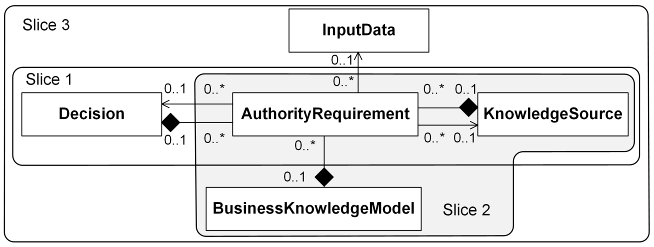
7.2.2. Evaluation of the Visual Expressiveness

Visual expressiveness [55] is heavily used in the scientific community for the evaluation of a modeling language's notation (see e.g., [30, 13]). The metric itself builds upon Bertins visual alphabet [5], visualized in figure 20b. The survey applies only the *visual expressiveness* metric because: (i) it is a non-ambiguous metric which is free of subjective perceptions of the evaluator, and (ii) most of the specifications do not have strict notation guidelines impeding an evaluation against the modeling language's purpose or semantics.

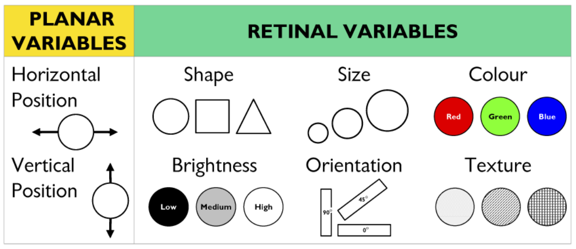
Bertins visual alphabet encompasses eight variables that equally add to the visual expressiveness of a modeling language. The more variables considered by a modeling language notation, the higher the visual expressiveness. A higher visual expressiveness is fostering perceptual discriminability of modeling constructs [55], i.e., constructs

Table 7
Usage of the notation specification techniques ((y)es=used, (n)o=not used) and specified aspects

	Notation Samples	Natural Language	Coordinate System	Used techniques	Dynamic Notation	Alternative Notation	Conformance Level
ArchiMate	y	y	n	2	n	y	y
BPMN	y	y	n	2	y	y	y
CMMN	y	y	n	2	y	n	y
DMN (DRD)	y	y	n	2	n	y	y
IFML	y	n	n	1	n	y	y
LML	y	n	n	1	n	y	n
OPM	y	y	n	2	y	n	y
S2ML	y	y	n	2	n	y	n
UML	y	y	n	2	y	y	y
URN	y	y	y	3	y	y	n
VDML	y	y	n	2	y	y	y
Used by	11/11	9/11	1/11		6/11	9/11	8/11



(a) *AuthorityRequirement* connector type specification by a connector type in DMN [63, p. 56]



(b) Bertins visual alphabet - taken from [55, p. 761]

Figure 20: Decision Modeling Notation (DMN) metamodel and Bertins visual alphabet

Table 8
Visual expressiveness of the surveyed modeling language notations

	Visual Expressiveness	Used Variables [5]
ArchiMate	4	Color, Shape, Positions (horizontal/vertical)
BPMN	4	Brightness, Shape, Positions (horizontal/vertical)
CMMN	3	Shape, Positions (horizontal/vertical))
DMN (DRD)	3	Shape, Positions (horizontal/vertical)
IFML	4	Brightness, Shape, Positions (horizontal/vertical)
LML	2	Positions (horizontal/vertical)
OPM	4	Color, Shape, Positions (horizontal/vertical)
S2ML	2	Positions (horizontal/vertical)
UML	4	Brightness, Shape, Positions (horizontal/vertical)
URN	5	Size, Brightness, Shape, Positions (horizontal/vertical)
VDML	3	Shape, Positions (horizontal/vertical)

should be easily distinguishable from one another. The eight variables are self-describing to a certain degree - for more details please see [55, 5]. Theoretically, the maximal visual expressiveness is 8, the lowest is 0. However, all surveyed languages are considered conceptual modeling languages that employ a diagrammatic representation. Consequently, the positions variables are given for all languages. The visual alphabet seems to focus on object types, not on connector types - see for example the size or texture variable. Consequently, this survey focused the analysis on the visual expressiveness of metamodel object types.

Table 8 summarizes the results of evaluating the visual expressiveness which will be delineated in the following. ArchiMate has a visual expressiveness of 4. The used variables are *shape, color*, and *positions*. The specification [78, p. 10] explicitly defines that *"the use of color is left to the modeler. However, they can be used freely to stress certain aspects in models"*. This is interesting, as in the specification itself, colors are used to distinguish between the layers of the ArchiMate framework. While the layers represent one dimension of the ArchiMate framework, the structure represents the second dimension. The shape of the objects represents to which structure the corresponding metaelement belongs to. *Behaviour* objects are represented by round boxes, *active and passive* structure elements are represented by boxes with square corners, and motivation objects are *"usually denoted using boxes with diagonal corners"* [78, p. 18]. *Passive* structure objects are represented with a box with squared corners and an additional horizontal line. ArchiMate follows the philosophy that a different notation should be used for different model users. Therefore the specification introduces a view-pointing mechanism which allows to create user-specific notations. The standard notation introduced in the specification can be used - but it is not binding - see [78, p. 10].

BPMN has a visual expressiveness of 4 and uses the variables *brightness, shape*, and *positions*. BPMN uses different shapes for different specializations of object types, e.g., events are represented by a circle and activities are represented by a rectangle. The brightness is e.g. used to distinguish between events which are catching events, such events are not filled, and throwing events, such events are filled. Similarly to ArchiMate, most of the BPMN notation guidelines are recommendations and therefore optional. A certain conformance level for notation is introduced whereby the referenced notation guidelines are optional. For example, in section 2.2.3 of the specification [58, p. 8], it is specified that BPMN process diagrams *shall* use the introduced graphical elements and that there is *flexibility in size, color, line style, and text positions of graphical elements*. Additionally, section 7.5 of the specification [58, p. 39-45] contains rudimentary notation guidelines. For example, color is not defined for the modeling language but may be used for BPMN models. Also, the line style and the placement of labels can be decided *depending on the preference of the modeler or modeling tool vendor* [58, p. 39]. Strict notation guidelines are rare - e.g. it is described that *markers for throwing Events must have a dark fill* [58, p. 39] or that the name of a pool *must be separated from the contents of the pool by a single line* [58, p. 111]. The specification defines alternative notations - there are for example alternative notations for exclusive and event-based gateways. For the notation of e.g. events dynamic notation is used which depends on the value of *EventDefinition*.

The Case Management Model and Notation (CMMN) specification [60] introduces a specific notation conformance level whereby - similar to the BPMN - flexibility is retained in e.g., size, color and line style. Only the binding notation guidelines have to be fulfilled in order to create compliant tools. CMMN also uses a dynamic notation, e.g., the notation of the instances of the object type *Task* depends on attribute value *blocking*. CMMN has a visual expressiveness of 3 by using the variables shape and positions. The specification does not provide alternative notations.

The DMN specification [63] also specifies flexibility in the notation guidelines for size, color and other notational aspects. The notation itself is specified via samples and natural language, e.g., the name of instances of the object type *Decision* should be visualized inside the shape of the element, or instances of the *KnowledgeSource* metaelement have inter-alia three straight sides. The specification also uses alternative notations e.g. for the metaelement *InputData*. DMN does not employ any dynamic notation.

The IFML specification [59, p. 1] introduces a notation conformance level for tool vendors which allows standard-defined notation to be *created, read, updated, and deleted*. However, the notation is not described in a precise way: Unlike other specifications, no natural language notation guidelines are introduced. The notation is only introduced via a table comprising sample notations. Alternative notations are used e.g. for the *Event* object type. The visual expressiveness of the IFML notation specification is 4 as *brightness*, different *shapes*, and *positions* are used.

LML specifies [50] a very simple notation which aims for low complexity. The complete LML notation is specified by samples of box and line diagrams, hence, a visual expressiveness of 2 has been assessed. While a *common visualization* is introduced *other visualizations are allowed and encouraged as they aid in expressing the information, which is the real goal of any language visualizations* [50, p. 5]. LML introduces different samples of diagrams which contain different notations (see figures 19b and 19c for selected examples with different notations for the OR metaelement). It seems like the specification does not aim at defining unique notations. For example, the notation of the risk matrix, an object type in LML, is explicitly declared as optional in [50, p. 59].

The notation guidelines introduced in the Object Process Methodology (OPM) [34] are binding. OPM introduces specific tool conformance criteria. The notation is specified using samples as well as natural language notation guidelines. The OPM specification proposes a dynamic notation. For example, the boarder line changes based on an attribute value [34, p. 16-17]. The OPM has a visual expressiveness of 4 as it uses the variables *color*, *shape* and *positions*.

The System Structure Modeling Language (S2ML) uses samples for the specification of the notation. The modeling elements are represented via simple boxes. The visual expressiveness of the S2ML is 2 as only the *position* variables are used. Conformance levels and dynamic notations are not specified, however, S2ML comes with an alternative notation.

The notation of the Unified Modeling Language (UML) specification [61] is binding and introduces a specific notation conformance level. Alternative notations are used, e.g. two different notations are introduced for the object

type *Actor* in Use-Case diagrams. Furthermore, UML uses dynamic notations. For example, the metaelement *PackageImport* is used as connector type and has an attribute *visibility*. Depending on the value of this attribute the notation of the connector changes: if the attribute has the value *public* the connector is annotated with the label *import*, otherwise it is annotated with the label *access*. UML is the only surveyed specification which introduces a precise description of the visualization of the labels. An example of the label of the metaelement *extension point* of Use-Case Diagrams is shown in the following [61, p. 642]:
ExtensionPoint is denoted by a text string within the UseCase oval symbol according to the syntax below:

```
<extension point> ::= <name> [: <explanation>]
```

The notation of the URN specification [35] has the highest visual expressiveness surveyed - value 5. URN introduces binding notation guidelines. If a binding notation is not given for an element, tool vendors have to define ways how to create, access and modify instances of these metaelements - e.g. by using a property window. The URN notation is specified with a high precision: Notation samples are used as well as natural language notation guidelines. Further, precise specifications of rendering objects by introducing a coordinate system as shown in figure 19d are given. Here it is precisely defined how the x and y coordinates as well as width and height are rendered. Like most of the other specifications, the URN introduces alternative notations e.g. for the metaelement *actorRef*.

The notation guidelines of the Value Delivery Modeling Language (VDML) specification [62] are binding. The VDML specification introduces alternative notations where two instances of an object type are represented differently in two different model types aka diagrams. The notation uses the variables *shape* and *positions*.

7.3. Further Observations

The preceding sections covered the results of evaluating the visual expressiveness of the modeling language notations. In addition, an evaluation of the visual expressiveness of the notation of the visual metamodels within the specifications was performed. It can be concluded that the notation of visual metamodels is currently not very expressive. All language specifications of the survey that use visual metamodels revert to simple box-and-line diagrams. An exception is the ArchiMate specification which uses different background colors for different metaelements in visual metamodels.

It is interesting to note, that the language notations have not been considerably revised in the last decades. Although the syntax for almost all surveyed languages has expanded greatly - e.g., the increase of metaelements in ArchiMate or BPMN, or the integration of the MetaObject Facility as a common meta-metamodel for the UML - the notation is still very simplified and minimalistic. With the uprise of advanced modeling editors as well as virtual and augmented reality novel opportunities for modeling language notations arise. Future research should invest in improving the existing notations - not only but also with respect to visual expressiveness.

Another interesting aspect found is that 9 out of the 11 surveyed specifications use alternative notations. Most of them even introduce them without clarifying the underlying rational for having multiple, alternative notations for one modeling construct. This is in total disrespect of the principle of semiotic clarity (symbol redundancy). Generally, it seems that the standards neglect most of the research on principles for designing effective visual notations by Moody [55]. Moreover, it is interesting that color, which is very good distinguished by human beings, is only used by 2 out of the 11 surveyed specifications.

All surveyed specifications except the LML use combined elements. Thereby, elements are positioned on the top of each/in each other. For example events in the BPMN which can be placed on the boarder of activities, or groups in ArchiMate which can encompass other ArchiMate objects.

8. Language Interoperability and Extension Specification

The specifications do not only contain aspects related to the modeling language itself. All of them introduce further language aspects such as *execution semantics, serialization formats, views* and *profiling* as summarized in table 9. These language aspects are often described in supplementary documents. For example the Open Group released a separate document for an ArchiMate file exchange format[13]. The Object Management Group (OMG) released for the Value Delivery Modeling Language (VDML) specification [62] additional serialization formats in a separate document. In the following, only the modeling language specification documents are surveyed in order to be consistent with the rest of the paper.

[13]https://www2.opengroup.org/ogsys/catalog/C174, last accessed: 2019-01-18

Execution Semantics. [76] defines the semantics which describe how models can be executed as execution semantics. Execution semantics is specified in all surveyed specifications using natural language - usually on a high level of abstraction. The Business Process Model and Notation (BPMN) specification [58] additionally specifies a token mechanism - similar to tokens in PetriNets. Also, the mapping of BPMN to Business Process Execution Language (BPEL) which is introduced in the specification helps to understand how BPMN models can be executed even if there are some issues as shown in [66, 18]. The Case Management Model and Notation (CMMN) specification [60] uses state diagrams in order to define the execution semantics. The Decision Modeling Notation (DMN) [63] specifies the execution semantics by referencing to expressions of the FEEL expression language. System Structure Modeling Language (S2ML) models are a visualization of the S2ML text - a mapping between text and visualization exists. The Unified Modeling Language (UML) execution semantics is specified by referring to the token concept, e.g. for activity diagrams. The OMG defined with the fUML [76] an executable subset of the UML language.

Serialization Format. Seven out of the 11 surveyed modeling languages specify a serialization format. The BPMN and the **URN** specifications [58, 35] introduce a separate XML schema. Interaction Flow Modeling Language (IFML), UML, BPMN and CMMN introduce a metamodel for the model interchange with OMG's Diagram Interchange[14] approach. The S2ML uses the modeling language as a visual representation for the textual S2ML - so S2ML models are serialized to S2ML text. The Object Process Methodology (OPM) specification describes a mapping to the textual language Object-Process language. The DMN provides a mapping of decision tables to FEEL - a serialization format for the Decision Requirements Diagram (DRD) was not identified. ArchiMate, Lifecycle Modeling Language (LML), and VDML do not specify a serialization format.

Views. Views in conceptual modeling are used to emphasize certain aspects of (large) models while omitting others [7]. Two language specifications introduce such a view mechanism to decompose an overarching modeling language into smaller views. The ArchiMate specification [78] introduces a view mechanism which conforms to the ISO/IEC 42010 standard[15]. Thereby, views are considered as mechanisms aiming to offer only the relevant information of an enterprise architecture to specific stakeholders. Viewpoints govern views and determine *"conventions for constructing, interpreting and analyzing the view"* [78, p. 106]. The specification [78] introduces a two-stepped process of creating ArchiMate viewpoints: (i) First, relevant metaelements of the ArchiMate standard have to be selected. (ii) The selected metaelements have to be visualized to the suggested user of the view. Therefore a complete new notation can be specified using ArchiMate's profiling mechanism. An example of a viewpoint creation is introduced in the appendix of the specification but technical details are missing. The IFML specification [59] introduces a specific object type *ViewPoint* which acts as container for elements which are relevant for a certain view. An example is the metaelement *InteractionFlowModel* which contains all model elements while the metaelement *ViewPoint* contains model elements relevant for a certain view.

The DMN specification [63] introduces a view mechanism which is used to hide information - see section 6.2.4 of the specification [63, p. 41]. Technical details how such a view is realized are missing. Similarly, OPM [34] introduces with the *zoom-in* and *zoom-out* models a view mechanism without technical details or predefined metamodel elements. As the view mechanisms of these two specifications are not described in more detail we neglected them in table 9.

All surveyed specifications that introduce views omit all technical details and also disregard consistency between the views (cf. [2, 38, 9]). This is a major deficit and limits applicability by the modelers and also hinders efficient development of modeling tools.

Profiling. The UML specification [61] introduces the well known *UML profiling* approach for extending the language with *stereotypes*. Also, the ArchiMate specification [78] uses a profiling mechanism - thereby attributes can be introduced as well as a notation. According to section 15.2 of the ArchiMate specification [78, p. 110], profiles may additionally introduce a new notation using the <<profile name>> label - the default notation is the one of the UML stereotypes. In the other specifications, no explicit profiling approach was identified.

9. Discussion of Key Findings

The following discussion is structured in two parts: The first part presents a retrospective by discussing the major challenges observed in current visual modeling language specifications. The second part then provides a discussion on the validity of the conducted survey.

[14]Diagram Interchange was replaced by the Diagram Definition Specification - http://www.omg.org/spec/DD/1.1/PDF/, last accessed: 2019-01-18

[15]https://www.iso.org/standard/50508.html, last accessed: 2019-01-11

Table 9
Specification of language interoperability and extension ((y)es=supported, (n)o=unsupported)

	Execution Semantics	Serialization Format	View	Profiling
ArchiMate	text	n	y	y
BPMN	text, token, mapping to BPEL	y	n	n
CMMN	text, state diagrams	y	n	n
DMN (DRD)	text, mapping to FEEL	n	n	n
IFML	text	y	y	n
LML	text	n	n	n
OPM	text	y	n	n
S2ML	text, mapping to S2ML text	y	n	n
UML	text, token	y	n	y
URN	text	y	n	n
VDML	text	n	n	n
Used by	11/11	7/11	2/11	2/11

9.1. Major Challenges Surveyed in Modeling Language Specifications

The findings reported previously are in the following condensed into a set of major challenges.

Heterogeneity of specifications and specification techniques. The structure and the techniques employed to specify a modeling language comprehensively are very heterogeneous (see tables 4, 5, 6, and 7 for an overview). Consequently, readers first need to have a decent understand of the general structure of the specification as well as the specification techniques before being able to actually comprehend the specification. One observation is that all Object Management Group (OMG) maintained specifications use a similar style and a similar structure for the first sections - preface and scope. However, no two surveyed OMG specifications use either identical metamodel concepts or techniques for specifying object types and connector types. The only remarkable similarity among all OMG specifications is that they use connector types for specifying connector types. The connectors (i.e., relations) in the surveyed OMG metamodels represent reference attributes.

The precision for specifying language constructs is heterogeneous, too. For example, when considering the specification of constraints, Unified Modeling Language (UML) and Interaction Flow Modeling Language (IFML) use Object Constraint Language (OCL) while all other surveyed specifications only use informal natural language. The labels used in notations are defined in the UML using Extended Backus-Naur form while the other specifications only use examples or/and natural language.

Incompleteness. In all surveyed specifications incompleteness of visual metamodels was attested. For a comprehensive syntax specification, the additional natural language description or OCL constraints have to be considered. Especially for large specifications such as the UML [61] this leads to a considerable effort for both, the ones creating and maintaining a specification, and for readers aiming to comprehend the specification. Some incompleteness of visual metamodels result from the limited expressiveness of currently used metamodel specification techniques. Here is where additional specification of e.g. constraints is inevitable (see section 6.3.5). What is important that in such situations one would expect that the specifications are explicit in what they specify in visual metamodels and what using amendments. However, this cannot be confirmed by the survey at hand.

This survey identified some incompletenesses of metamodels because the maintainers seem to keep the metamodels and the specifications simple. An example is the ArchiMate specification [78] that often only shows excerpts of the metamodel without explicitly stating what has been left out. One problematic and recurring example is that the metamodels, in this case the business layer metamodel as visualized in figure 5b, visually encode, that *Business Interfaces* can be connected to *Business Services* by an *assigned to* connector. However, when looking at Appendix B of the specification [78, p. 120ff], one recognizes, that *association, flow, serving, and triggering* connectors are also allowed. Thus, the visual specification is inconsistent, Table 6 even shows, that seven out of the 11 surveyed specifications actually do not specify connector types in their visual metamodels at all.

A different obstacle of incompleteness refers to the mechanisms & algorithms, and the modeling procedure of a modeling language (cf. [39]). The Object Process Methodology (OPM) specification [34] is the only surveyed specification which introduces modeling guidelines. With the increase of metamodel-size and, as a consequence, the size of specification documents, maintaining institutions should deliberate some time to e.g., design a learning path or reflect on how a modeling language should be applied.

- **Element Name**
- **Syntax**
  - Attributes
  - Relationships
  - Constraints
- **Notation**
- **Semantics**
- **Notation Example**

(a) Structured Template used e.g., in URN [35]

- **Element Name**
- **Description of Element**
- **Attribute Table**
- **Notation Example**

(b) Semi-structured Template used e.g., in BPMN [58]

Figure 21: Used templates for describing metaclasses of metamodels in specification documents

No referral to a meta-metamodel. Some specifications do not refer to a meta-metamodel when introducing the abstract syntax of a modeling language (i.e., a metamodel) which also impairs understanding. Especially when thinking of languages that shall enable extension, such aspects are vital. This challenge is even more serious, as the hierarchical structure of modeling languages is well studied and established, both in the academic conceptual modeling community [75] and in the industrial metamodeling platform concepts [39, 42].

No uniform specification of connector types. As table 5 indicates, the survey showed a great diversity in how connector types are visually specified. One specification, on average, employs three different specification techniques without clearly introducing them beforehand. This inconsistent usage compared with the lack of introduction to the reader severely hampers comprehension. Employing a unified technique for the specification of connector types in visual metamodels should simplify comprehension. Connector types in metamodels should be indicated by special labels or symbols, easing to distinguish them visually from object types.

No established specification structure. This survey revealed recurring structures in the form of specification templates that have been used (see figure 21). Two basic types of templates can be distinguished: *structured and semi-structured templates*. The former distinguishes between characteristics of the elements such as semantics, constraints or notation while the latter mixes the specifications of e.g. constraints, notation guidelines and semantics.

Specification maintaining institutions should harmonize at least their own specifications. This may comprise a uniform template for the specification of the elements of the visual metamodel. One of the structured templates used in the URN specification [35] (see figure 21a) might be a good starting point. Also the scope of specifications could be harmonized by publishing separate documents for additional language components such as serialization formats or execution semantics (see table 9)) which are only useful for a specific group of readers. This focuses the core language specifications on their essential parts and should therefore contribute to faster comprehension by its addressees.

9.2. Validity of the Survey

As any empirical study, this study comes with some threats to validity. The employed search strategy is a relevant threat: Existing specifications probably have been missed, especially regarding less prominent specifications which are not maintained by organizations such as OMG. To ensure that the survey encompasses all widely used specifications our search strategy included explicitly the specification database of OMG and the Open Group. Furthermore, the exclusion of specifications with restricted or monetized access as well as the time constraint might represent a threat.

The results presented in section 8 are limited to some extent because the survey investigated only the core specification documents - additional material provided by maintaining institutions or researchers was excluded. However, the section shows that the scope of specifications is very heterogeneous.

While conducting this survey, preliminary results have been discussed between all authors and with other researchers in order to identify missing important aspects or to streamline observations and terminology. The survey at hand is the result of several iterations.

10. Conclusion and Further Research

Visual modeling languages play an important role in information systems analysis and design [51, 71]. Usually, such languages are introduced in large specification documents. Such specifications are vital for e.g., the utilization of the language by modelers, the domain-specific adaptation/extension by researchers, and the development of conforming modeling tools by vendors. With the rise of domain-specific modeling languages (see [40, 10] for an overview) and model-driven development approaches (cf. [54, 4]), more and improved specifications will be needed in the future.

Albeit the aforementioned, only little is known about how modeling languages are specified and how to create a comprehensive and consistent specification. This paper surveyed 11 currently widely used modeling language specifications. The results show that current specifications have little similarity with respect to the metamodel concepts being specified and the techniques being used for their specification. This is probably due to the absence of: 1) a meta-specification which contains guidelines and structures for describing specifications, and 2) a meta-metamodel that which introduces the language used for metamodel specifications. This survey aims to contribute a foundation towards developing such a meta-specification and a meta-metamodel in the future. In this respect, empirical research needs to investigate the effects of the identified specification techniques on comprehension (cf. [73]) and ease of use.

We are confident that our research can have a strong impact by positively influencing the creation of new, and future revision of existing modeling language specifications. This ultimately leads to an improved utilization of the modeling methods in information systems analysis and design.

References

[1] Allaki, D., Dahchour, M., En-Nouaary, A., 2017. Managing inconsistencies in UML models: A systematic literature review. JSW 12, 454–471.

[2] Awadid, A., Bork, D., Karagiannis, D., Nurcan, S., 2018. Toward generic consistency patterns in multi-view enterprise modelling, in: Twenty-Sixth European Conference on Information Systems (ECIS'2018), Portsmouth, UK.

[3] Azevedo, C.L., Iacob, M.E., Almeida, J.P.A., van Sinderen, M., Pires, L.F., Guizzardi, G., 2015. Modeling resources and capabilities in enterprise architecture: A well-founded ontology-based proposal for archimate. Information systems 54, 235–262.

[4] Basin, D., Doser, J., Lodderstedt, T., 2006. Model driven security: From uml models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. 15, 39–91.

[5] Bertin, J., 2010. Semiology of Graphics - Diagrams, Networks, Maps. ESRI.

[6] Blouin, A., Moha, N., Baudry, B., Sahraoui, H., Jézéquel, J.M., 2015. Assessing the use of slicing-based visualizing techniques on the understanding of large metamodels. Information and Software Technology 62, 124–142.

[7] Bork, D., 2015. Using conceptual modeling for designing multi-view modeling tools, in: 21st Americas Conference on Information Systems, AMCIS 2015, Puerto Rico, August 13-15, 2015.

[8] Bork, D., 2018. Metamodel-based Analysis of Domain-specific Conceptual Modeling Methods, in: Buchmann, R.A., Karagiannis, D., Kirikova, M. (Eds.), IFIP Working Conference on The Practice of Enterprise Modeling, Springer. pp. 172–187.

[9] Bork, D., Buchmann, R.A., Karagiannis, D., 2015. Preserving multi-view consistency in diagrammatic knowledge representation, in: Zhang, S., Wirsing, M., Zhang, Z. (Eds.), Knowledge Science, Engineering and Management - 8th International Conference, KSEM 2015, Proceedings, Springer. pp. 177–182.

[10] Bork, D., Buchmann, R.A., Karagiannis, D., Lee, M., Miron, E.T., 2019. An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem. Communications of the Association for Information Systems 44, pp. 673–697.

[11] Bork, D., 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.

[12] Bork, D., Karagiannis, D., Pittl, B., 2018a. How are Metamodels Specified in Practice? Empirical Insights and Recommendations, in: Twenty-fourth Americas Conference on Information Systems, pp. 1–10.

[13] Bork, D., Karagiannis, D., Pittl, B., 2018b. Systematic Analysis and Evaluation of Visual Conceptual Modeling Language Notations, in: 2018 12th International Conference on Research Challenges in Information Science (RCIS), IEEE. pp. 1–11.

[14] Broy, M., Cengarle, M.V., 2011. Uml formal semantics: lessons learned. Software & Systems Modeling 10, 441–446.

[15] Buchmann, R.A., Karagiannis, D., 2016. Enriching linked data with semantics from domain-specific diagrammatic models. Business & Information Systems Engineering 58, 341–353.

[16] Costagliola, G., Lucia, A.D., Orefice, S., Polese, G., 2002. A classification framework to support the design of visual languages. J. Vis. Lang. Comput. 13, 573–600.

[17] De Lara, J., Guerra, E., Cuadrado, J.S., 2013. Reusable abstractions for modeling languages. Information Systems 38, 1128–1149.

[18] Debevoise, T., 2014. Execution semantics, in: White Paper, Black Pearl Development. OMG, pp. 1–25.

[19] Di Rocco, J., Di Ruscio, D., Iovino, L., 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.

[20] Dijkman, R., Dumas, M., Van Dongen, B., Käärik, R., Mendling, J., 2011. Similarity of business process models: Metrics and evaluation. Information Systems 36, 498–516.

[21] Dijkman, R.M., Dumas, M., Ouyang, C., 2008. Semantics and analysis of business process models in bpmn. Information and Software technology 50, 1281–1294.

[22] Döller, V., 2018. Formal Semantics for Conceptual Modeling Languages based on Model Theory, in: Proceedings of the Doctoral Consortium Papers presented at the 11th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling (PoEM 2018), pp. 61–73.

[23] Dori, D., 2016. Model-Based Systems Engineering with OPM and SysML. Springer.

[24] Drawehn, J., Kochanowski, M., Kötter, F., 2014. Business process management tools 2014: Marktüberblick;[Überblick über die verfügbaren Werkzeuge für das Geschäftsprozessmanagement im deutschsprachigen Raum]. Fraunhofer Verlag.

[25] Eclipse, 2019. Eclipse modeling framework (emf). Eclipse URL: https://www.eclipse.org/modeling/emf/. accessed on 2019-01-23.

[26] Fayoumi, A., Loucopoulos, P., 2016. Conceptual modeling for the design of intelligent and emergent information systems. Expert Systems with Applications 59, 174–194.

[27] Fellmann, M., Bittmann, S., Karhof, A., Stolze, C., Thomas, O., 2013. Do we need a standard of EPC modelling? the state of syntactic, semantic and pragmatic quality, in: Enterprise Modelling and Information Systems Architectures: Proceedings of the 5th International Workshop on Enterprise Modelling and Information Systems Architectures, EMISA 2013, St. Gallen, Switzerland, September 5-6, 2013, pp. 103–116.

[28] Fleck, M., Troya, J., Wimmer, M., 2016. Towards generic modularization transformations, in: Companion Proceedings of the 15th International Conference on Modularity, ACM. pp. 190–195.

[29] France, R., Evans, A., Lano, K., Rumpe, B., 1998. The uml as a formal modeling notation. Computer Standards & Interfaces 19, 325–334.

[30] Genon, N., Heymans, P., Amyot, D., 2010. Analysing the cognitive effectiveness of the bpmn 2.0 visual notation, in: International Conference on Software Language Engineering, Springer. pp. 377–396.

[31] Ghiran, A.M., Buchmann, R.A., Karagiannis, D., 2018. Towards a framework of techniques for enabling semantics-driven secondary notation in conceptual models, in: 2018 12th International Conference on Research Challenges in Information Science (RCIS), IEEE. pp. 1–6.

[32] Henderson-Sellers, B., Ralyté, J., 2010. Situational method engineering: State-of-the-art review. J. UCS 16, 424–478.

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

[34] International Organization for Standardization, 2014. Object process methodology (opm) specification version 522. URL: https://www.iso.org/standard/62274.html. accessed on 2019-05-12.

[35] International Telecommunications Union, 2018. User requirements notation (urn) specification version z.151. URL: https://www.itu.int/rec/T-REC-Z.151-201810-I/en. accessed on 2019-05-13.

[36] Jannaber, S., Karhof, A., Riehle, D.M., Thomas, O., Delfmann, P., Becker, J., 2016. Invigorating event-driven process chains - towards an integrated meta model for EPC standardization, in: Modellierung 2016, 2.-4. März 2016, Karlsruhe - Workshopband, pp. 13–22.

[37] Karagiannis, D., Buchmann, R.A., 2016. Linked open models: extending linked open data with conceptual model information. Information Systems 56, 174–197.

[38] Karagiannis, D., Buchmann, R.A., Bork, D., 2016a. Managing consistency in multi-view enterprise models: an approach based on semantic queries, in: 24th European Conference on Information Systems, ECIS 2016, Istanbul, Turkey, June 12-15, 2016, p. Research Paper 53.

[39] Karagiannis, D., Kühn, H., 2002. Metamodelling platforms, in: E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings, p. 182.

[40] Karagiannis, D., Mayr, H.C., Mylopoulos, J. (Eds.), 2016b. Domain-Specific Conceptual Modeling, Concepts, Methods and Tools. Springer.

[41] Karhof, A., Jannaber, S., Riehle, D.M., Thomas, O., Delfmann, P., Becker, J., 2016. On the de-facto standard of event-driven process chains: Reviewing EPC implementations in process modelling tools, in: Modellierung 2016, 2.-4. März 2016, Karlsruhe, pp. 77–92.

[42] Kelly, S., Lyytinen, K., Rossi, M., 1996. Metaedit+ a fully configurable multi-user and multi-tool case and came environment, in: International Conference on Advanced Information Systems Engineering, Springer. pp. 1–21.

[43] Kern, H., Hummel, A., Kühne, S., 2011. Towards a comparative analysis of meta-metamodels, in: Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11, ACM. pp. 7–12.

[44] Kitchenham, B., Brereton, P., 2013. A systematic review of systematic review process research in software engineering. Information & Software Technology 55, 2049–2075.

[45] Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering, in: Technical report, Ver. 2.3 EBSE Technical Report. EBSE. sn, pp. 1–65.

[46] Kitchenham, B.A., Dyba, T., Jorgensen, M., 2004. Evidence-based software engineering, in: Proceedings of the 26th international conference on software engineering, IEEE Computer Society. pp. 273–281.

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

[48] Ko, R.K.L., Lee, S.S.G., Lee, E.W., 2009. Business process management (BPM) standards: a survey. Business Proc. Manag. Journal 15, 744–791.

[49] Lalioti, V., Loucopoulos, P., 1994. Visualization of conceptual specifications. Information Systems 19, 291–309.

[50] Lifecycle Modeling, 2015. Lifecycle modeling language (lml) specification version 1.1. URL: http://www.lifecyclemodeling.org/specification/. accessed on 2019-02-04.

[51] Lyytinen, K., Welke, R., 1999. Guest editorial: Special issue on meta-modelling and methodology engineering. Information Systems 2, 67–69.

[52] Ma, Z., He, X., Liu, C., 2013. Assessing the quality of metamodels. Frontiers of Computer Science 7, 558–570.

[53] Marriott, K., Meyer, B., Wittenburg, K.B., 1998. A survey of visual language specification and recognition, in: Visual language theory. Springer, pp. 5–85.

[54] Mattsson, A., Fitzgerald, B., Lundell, B., Lings, B., 2012. An approach for modeling architectural design rules in uml and its application to embedded software. ACM Trans. Softw. Eng. Methodol. 21, 10:1–10:29.

[55] Moody, D., 2009. The 'physics' of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering 35, 756–779.

[56] Mylopoulos, J., 1992. Conceptual modelling and telos. Conceptual Modelling, Databases, and CASE: an Integrated View of Information System Development, New York: John Wiley & Sons , 49–68.

[57] Nissen, H.W., Jarke, M., 1999. Repository support for multi-perspective requirements engineering. Information Systems 24, 131–158.

[58] Object Management Group, 2014. Business process model and notation (bpmn) specification version 2.02. URL: http://www.omg.org/spec/BPMN/2.0.2/PDF/. accessed on 2019-05-13.

[59] Object Management Group, 2015. Interaction flow modeling language (ifml) specification version 1.0. URL: http://www.omg.org/spec/IFML/1.0/PDF/. accessed on 2019-05-13.

[60] Object Management Group, 2016. Case management model and notation (cmmn) specification version 1.1. URL: http://www.omg.org/spec/CMMN/1.1/PDF/. accessed on 2019-05-13.

[61] Object Management Group, 2017. Unified modeling langague specification (uml) specification 2.5.1. URL: https://www.omg.org/spec/UML/2.5.1/PDF/. accessed on 2017-12-05.

[62] Object Management Group, 2018. Value delivery modeling language (vdml) specification version 1.1. URL: https://www.omg.org/spec/VDML/1.1/PDF. accessed on 2019-05-13.

[63] Object Management Group, 2019. Decision modeling notation (dmn) specification version 1.2. URL: https://www.omg.org/spec/DMN/1.2/PDF. accessed on 2019-05-13.

[64] OpenAltaRica, 2015. System structure modeling language (s2ml) specification version 1.0. URL: https://hal.archives-ouvertes.fr/hal-01234903/document. accessed on 2019-05-13.

[65] Paige, R.F., Brooke, P.J., Ostroff, J.S., 2007. Metamodel-based model conformance and multiview consistency checking. ACM Trans. Softw. Eng. Methodol. 16.

[66] Recker, J.C., Mendling, J., 2006. On the translation between bpmn and bpel: Conceptual mismatch between process modeling languages, in: The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium, Namur University Press. pp. 521–532.

[67] Riehle, D.M., Jannaber, S., Karhof, A., Thomas, O., Delfmann, P., Becker, J., 2016. On the de-facto standard of event-driven process chains: How EPC is defined in literature, in: Modellierung 2016, 2.-4. März 2016, Karlsruhe, pp. 61–76.

[68] Ritter, D., May, N., Rinderle-Ma, S., 2017. Patterns for emerging application integration scenarios: A survey. Information Systems 67, 36–57.

[69] Rosa, M.L., Van Der Aalst, W.M., Dumas, M., Milani, F.P., 2017. Business process variability modeling: a survey. ACM Computing Surveys (CSUR) 50, 2.

[70] Rosemann, M., 2006. Potential pitfalls of process modeling: part A. Business Process Management Journal 12, 249–254.

[71] Sandkuhl, K., Fill, H.G., Hoppenbrouwers, S., Krogstie, J., Matthes, F., Opdahl, A., Schwabe, G., Uludag, Ö., Winter, R., 2018. From expert discipline to common practice: a vision and research agenda for extending the reach of enterprise modeling. Business & Information Systems Engineering 60, 69–80.

[72] Scheer, A.W., Hars, A., 1992. Enterprise modeling: Basis for information systems design, in: Analyzing and Modeling Data and Knowledge. Springer, pp. 217–224.

[73] Snook, C.F., Harrison, R., 2004. Experimental comparison of the comprehensibility of a Z specification and its implementation in Java. Information and Software Technology 46, 955–971.

[74] Spencera, R., Teorey, T.J., Hevia, E., 1990. ER standards proposal, in: Proceedings of the 9th International Conference on Entity-Relationship Approach (ER'90), 8-10 October, 1990, Lausanne, Switzerland., pp. 405–412.

[75] Strahringer, S., 1998. Ein sprachbasierter metamodellbegriff und seine verallgemeinerung durch das konzept des metaisierungsprinzips., in: Modellierung, pp. 15–20.

[76] Tatibouet, J., Cuccuru, A., Gérard, S., Terrier, F., 2014. Formalizing execution semantics of UML profiles with fuml models, in: Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings, pp. 133–148.

[77] Thalheim, B., 2013. Entity-relationship modeling: foundations of database technology. Springer Science & Business Media.

[78] The Open Group, 2017. Archimate specification version 3.0.1. URL: https://publications.opengroup.org/c179. accessed on 2019-05-13.

[79] Tiwari, S., Gupta, A., 2015. A systematic literature review of use case specifications research. Information and Software Technology 67, 128–158.

[80] Van Der Aalst, W.M., 2013. Business process management: a comprehensive survey. ISRN Software Engineering 2013.

[81] Williams, J.R., Zolotas, A., Matragkas, N.D., Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A., 2013. What do metamodels really look like? Eessmod@ Models 1078, 55–60.