Software and Systems Modeling (2022) 21:1411–1435
https://doi.org/10.1007/s10270-022-00986-2

SPECIAL SECTION PAPER

Formalizing the four-layer metamodeling stack with METAMORPH: potential and benefits

Victoria Döller[1]

Received: 12 March 2021 / Revised: 12 January 2022 / Accepted: 4 February 2022 / Published online: 25 March 2022
© The Author(s) 2022

Abstract

Enterprise modeling deals with the increasing complexity of processes and systems by operationalizing model content and by linking complementary models and languages, thus amplifying the model value beyond mere comprehensible pictures. To enable this amplification and turn models into computer-processable structures, a comprehensive formalization is needed. This paper presents the formalism METAMORPH based on typed first-order logic and provides a perspective on the potential and benefits of formalization that arise for a variety of research issues in conceptual modeling. METAMORPH defines modeling languages as formal languages with a signature Σ—comprising object types, relation types, and attributes through types and function symbols—and a set of constraints. Four case studies are included to show the effectiveness of this approach. Applying the METAMORPH formalism to the next level in the hierarchy of models, we create `M2FOL`, a formal modeling language for metamodels. We show that `M2FOL` is self-describing and therefore complete the formalization of the full four-layer metamodeling stack. On the basis of our generic formalism applicable to arbitrary modeling languages, we examine four current research topics—modeling with power types, language interleaving & consistency, operations on models, and automatic translation of formalizations to platform-specific code—and how to approach them with the METAMORPH formalism. This shows that the rich knowledge stack on formal languages in logic offers new tools for old problems.

Keywords Conceptual modeling · Metamodeling · Modeling language · Formal language · Predicate logic

1 Introduction

Enterprise modeling has proven instrumental in facing the challenges of increasing complexity and interdependences of processes and systems in the modern world. Research on enterprise modeling has yielded some highly specialized tools with value-adding mechanisms like information querying, simulation, and transformation [6,27,59]. The nature of models has evolved from a visual representation of information to an exploitable knowledge structure [12]. Nevertheless, the European enterprise modeling community experiences that the potential of enterprise modeling is currently not fully utilized in practice and modeling is employed only by a limited group of experts. Therefore, in [59] a research agenda is formulated to establish "modeling for the masses" (MftM) and broadcast its benefits also to non-experts.

Communicated by Dominik Bork and Janis Grabis.

✉ Victoria Döller
victoria.doeller@univie.ac.at

1 Research Group Knowledge Engineering, Faculty of Computer Science, University of Vienna, Vienna, Austria

1.1 A need for formalization

To amplify the value of models as exploitable knowledge structures, we need to employ the capacities of machines. Models have to be implemented on a computer to operationalize their content beyond comprehensible pictures on a sheet of paper. Therefore, they must be expressed in a precise and unambiguous way. A machine-readable representation of a model in turn benefits from an elaborated formal foundation that offers means for model analysis, interoperability, and a sound basis for diverse mechanisms, as mentioned above.

Furthermore, also the utilization of modeling by non-experts as aspired to by the MftM movement implicitly benefits from a solid formal foundation. Although the initiators of the MftM movement mention that the formality of a model representation possibly hampers understandability, we argue that it is an implied need of several listed challenges and visions of conceptual modeling, respectively.

MftM requires the following:

1) Concerning the *stakeholder dimension* of challenges in enterprise modeling research [59, p. 234], the MftM initiators list computers as stakeholders. While the assumption that computers have interests on their own might be disputable, they indeed play an important role in the exploitation of model content and are model consumers and producers. This implies that models have to be formalized to make them computer-processable because computers do not understand semi-formal or unstructured models and language specifications [7].
2) The vision of models being not autotelic but *being a means to the operationalization of information* [59, p. 229] calls for value-adding functionality beyond mere graphics like analysis, reasoning, verification & validation, model transformation or simulation, functionality that preferably should become an integral part of a modeling language. This requires a specification of the syntax and operations that is formulated ideally implementation-independently and computer-understandably, i.e., formalized.
3) The vision of *local modeling practices that are globally integrative* [59, p. 229] calls for a common foundation and theory of models and modeling languages to enable the linking and merging of models in different domains with different semantics [32]. This vision of a global integrability especially points to the need for a generic, language-crossing foundation, a requirement going beyond a single language. This means that the need for formalization is not necessarily grounded in the language itself, which may be successful without formalization. It is the integrability in the big picture of models expressed with varying modeling languages, the reusability, and interoperability that brings about this need.

At the same time, MftM argues that overly formal modeling practices may hamper the understandability of models. We claim that this is not necessarily the case because a formalized model representation is an alternative way to denote the situation depicted in a graphical model representation with different merits. While the graphical representation is suitable to be consumed by humans, the formal representation resides in the background, offering opportunities for further exploitation.

The above requirements go even beyond a formalism in the sense of a precise and unambiguous specification. They reveal a need for a foundational formal structure complying with the inherent characteristics of conceptual modeling. This structure must comprise the relevant concepts of languages and provide an integrative foundation open for affiliation of any progression in conceptual modeling, may it be advanced concepts or sophisticated functionality. Only a formal foundation complying with the characteristics of conceptual modeling provides means for this mature integrability.

This is all the more essential in the light of the emergent importance of domain-specific modeling languages (DSMLs) [25] as well as increasing agility in the advancement and extension of established languages and methods [36]. The lack of a common way for formalizing DSMLs leads to divergent or lacking formal foundations limiting the opportunities to compare or link models. Frequently the big standards are extended for a specific domain, e.g., the extension of i* with security concepts constituting the modeling language Secure Tropos [50]. Therefore a common way of specifying the base languages as well as the extensions or modules is required. A silo-like formalization of the big standards is not sufficient as divergent base concepts of models and different underlying formal structures can impede a mutual interconnection and integration.

Another important building block for advancing the science of conceptual modeling is an exact and commonly applied method for specifying the metamodels of modeling languages. A survey conducted by Bork et al. showed that the specification documents of the standardized languages like UML and ArchiMate diverge in the concepts they consider as well as in the techniques they use to specify their visual metamodels [9]. Examples from recent scientific publications indicate that also in research on domain-specific languages, no common practice of metamodel specification is in use. Several contributions specify metamodels with UML class diagrams, declaring object types as classes and relation types as classes or named association arrows, e.g., [35,46,57,60,62]. Others simply define the object and relation types with box-and-line models devoid of an underlying language and rely on the intuitive understanding of the reader, e.g., [45,55]. This shows that although metamodels are models themselves and therefore subject of interest for enterprise modeling research no language for metamodels has been established yet. Nevertheless, when a language has to be implemented or executed, a precise and unambiguous definition of the metamodel is crucial [7].

1.2 Goal and requirements

According to the AMME framework of agile modeling method engineering shown in Fig. 1, the phase of formalization is pivotal in the lifecycle of a modeling language. Yet, there is no common procedure of how to formalize arbitrary modeling languages. Existing formalization approaches often restrict to a concrete application, domain or language, thereby limiting the reusability in other domains and languages. As the AMME lifecycle is meant as a generic procedure model for generating arbitrary modeling languages and methods, we need a formalism applicable to any domain

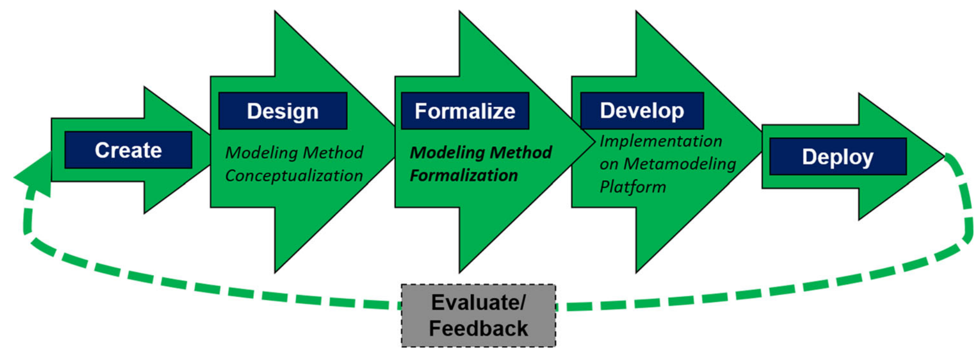

Fig. 1 The AMME lifecycle for agile modeling method engineering adapted from [36]

and language an engineer might come up with. The work at hand intends to close this gap and aims at building a bridge between the *Design* phase of collecting the relevant concepts and the *Develop* phase of transferring the final design to a metamodeling platform in the AMME lifecycle. Such a formalism must comply to the characteristics and structure of a modeling language to offer the required integrability of languages. For this reason, we raise the question: What exactly is a conceptual modeling language from a formal, structural point of view? The aim of the presented approach is to resolve this question step by step working toward an integrative formal foundation. We summarize the concrete requirements for the formalism as follows:

1) it has to be complete regarding the general building blocks of a language, 2) it must comply with the linguistic character of modeling languages, 3) it must be generic in a way that it admits the formalization of any language developed according to the four-layer metamodeling stack, an architecture often used in practice nowadays [1,11], and 4) it must provide an integrative formal foundation offering canonical tools for the advancements in conceptual modeling research.

In the work at hand, we present the generic and integrative formalism METAMORPH and demonstrate that in accordance with the requirements constituted above we can answer the question from above about the structural nature of modeling languages as follows: modeling languages can be defined as formal languages in the sense of logic. This means they comprise a signature Σ for the syntax and a set of constraints, for which we use first-order predicate logic. This paper extends our prior work [21] published at the PoEM 2020 conference about the definition of modeling languages, where we concretely stated how the core concepts of these languages can be expressed in logical terms. Predicate logic provides the construct of a Σ-structure, i.e., an interpretation of the signature, which is the canonical correspondent to the model conforming to a metamodel. Applying the METAMORPH formalism to the meta-language level results in M2FOL, a formal modeling language for metamodels. With M2FOL, we are capable of modeling the syntax of a language to be specified, to be more precise, the signature of the language according to the definition. The paper at hand extends the presented definition of formal modeling languages as well as the language M2FOL with the concept of multi-value attributes. We furthermore exemplify the potential and benefits inherent to the proposed formalism on a diverse range of research topics and demonstrate the opportunities this integrative foundation offers.

The rest of this paper is structured as follows: In Sect. 2, we give an overview of related work on formalization of metamodels and modeling languages. In Sect. 3, we introduce the METAMORPH formalism comprising the definition of formal modeling languages and models and concretize how the basic concepts of a language—object and relation types, attributes, specialization, and constraints—can be expressed in logical terms. We then use this definition in Sect. 4 to create M2FOL—a formal modeling language for metamodels—and outline its self-describing character. Given a metamodel specified with M2FOL we show how to algorithmically deduce the signature of the corresponding modeling language. After that we give in Sect. 5 an outlook to the potential and benefits of formal modeling languages. We present some ongoing research and approaches on modeling with power types, on how to interleave modeling languages, formally include operations on models into the language specification, and on the use of the formalization as a single point of specification processable by machines. In Sect. 6, we discuss the formalism with respect to the formulated goals and requirements and outline the agenda of the empirical evaluation that is currently being conducted.

2 Background and related work

2.1 Formalism vs. formal syntax

We begin by discussing the distinction between a *formalism* and a *formal way of specifying a language*. A formalism always gives rise to a formal specification. The converse, however, is not true. This can be compared to the concept of a graph and the unique and precise way of specifying a graph with an adjacent matrix. Each graph can be represented as a matrix, but a matrix per se does not provide the

semantics of graph theory. The same applies to specification techniques merely offering a formal syntax to describe modeling languages. These techniques provide a unique way of specification but lack the structure and semantics of the underlying components of conceptual modeling. A formal syntax may offer an expression for specifying the specialization of object types. Nevertheless, the expression does not accomplish the inherent semantics, i.e., the transfer of features of the supertype to the subtype. This behavior must be added to the syntax system by hand, although a suitable underlying structure would entail concepts for specialization. For this simple case, basic set theory would suffice to capture specialization via sets and containment automatically, extending functions defined on the superset to all subsets.

Our principal segregation of a formalism and a formal syntax distinguishes our goal from existing approaches like the meta-object facility (MOF) standard [14]. The MOF offers a concrete syntax or notation system for specifying modeling languages but no inherent theory and methods for the concepts to be described. Its specification is presented in an exhaustive natural language document without the intention to give a foundational structural theory. For this reason, it indeed fosters a common and generic way of specifying arbitrary metamodels (Req. 3) but lacks the benefits of a proper formalism. In Sect. 5, we will give an outlook to the potential of deploying a suitable formalism instead of a mere notation system, which will unveil the advantage of our formalism over the notation system of MOF.

Unlike the MOF standard, the language Z has a powerful mathematical foundation based on set theory and first-order logic and was designed for the formal specification of systems [54,61]. Z comprises several concepts, which are similar to elements in conceptual modeling, e.g., a schema in Z is similar to an object type in conceptual modeling. Nevertheless, the concepts differ heavily in their semantics. Schemata provide broad functionality, which is irrelevant for object types, while the aspect of the concrete model, the instance of a language, is not in the scope of a specification written in Z. Therefore, using Z for defining modeling languages requires hacking the semantics of concepts in Z. For this reason, Z does not fulfill the second postulated requirement, as it does not adequately reflect the characteristics of modeling languages and there is no obvious way how to create a modeling language specification with it.

In current research, the notion of formal languages in the sense of mathematical logic as underlying structure for modeling languages has been receiving increasing attention, as both comprise an alphabet or vocabulary and well-formedness rules [18,28,53,56,63]. Of course, not every formal language is a modeling language, but modeling languages form a subclass of all formal languages. In this paper, we want to concretize and work out this class of formal modeling languages and establish it as an integrative formal foundation of conceptual modeling.

2.2 Formalisms for concrete modeling languages

According to the Characterizing Conceptual Model Research (CCMR) framework [19] we are interested in contributions located in the dimension *Formalize* working on the level of *Conceptual Modeling Languages* and *Metamodeling Languages*. In this respect, we want to delineate our approach from the various attempts addressing the formalization of a specific modeling language. These attempts mostly aim at supporting a specific purpose or functionality and do not provide means to define arbitrary metamodels and languages. An example is the OSM-logic using typed predicate logic for object-oriented systems modeling with a focus on time-dependent system behavior [17]. Another example is the SAVE method for simulating IoT systems using the δ-calculus [16]. These specific formalizations may offer ideas suitable for being generalized to a generic approach but will not be comprehensively discussed here. However, as soon as there is a common practice of formally defining the ubiquitous concepts of modeling languages, these specific approaches can be constructed as reusable extensions and modules and be of value in a broader field of application.

2.3 Formalisms for ontologies and concept spaces

For a systematic positioning of our approach, we use the triptych allegory proposed by Mayr and Thalheim [47]. They define conceptual modeling as tripartite consisting of three dimensions: an *encyclopedic dimension* for codifying the individual cognitive concepts and notion in a commonly accepted encyclopedic structure like an ontology or concept space, a *language dimension* for the definition of language terms and valid expressions, and a *conceptual modeling dimension* in between as a link between term and encyclopedic structure. We are mainly interested in a formalization of the language dimension and acknowledge that in the encyclopedic dimension there also exist various attempts to formalization, like the KL-ONE family [10] and Description Logic [3]. Also, the formal system of a conceptualization of domains as basis for truthful modeling languages proposed by Guizzardi [29] has to be located in the encyclopedic dimension and has therefore to be distinguished from our goal. In this theory of ontologically-driven conceptual modeling fruitful for the objective of a domain-faithful grounding for modeling languages, the language dimension is an a-posteriori concept implicitly obtained from ontological considerations.

2.4 Formalisms for languages

When focusing on formalizations in the language dimension, the existing approaches can be categorized according to the underlying theory they use, which is mostly graph theory, set theory, or logic. All three of them offer concepts for the concrete structural behavior of the elements to be described. In the following we present examples illustrating the shortcomings of the former two and argue why logic provides the most canonical approach.

In the domain-specific language KM3 presented by Jouault and Bezivin [34] models are defined as directed multigraphs conforming to another model, the metamodel, itself a graph. Using this formalism, the authors define a self-describing metametamodel and deduce a domain-specific language to specify metamodels. This approach puts an emphasis on the graph-like box-and-line structure of models rather than on the linguistic aspects and has a narrow focus on software structure specification.

A system based on set theory is the formalization of Ecore and EMOF proposed by Burger [13, 2.3.2] which uses the formal description of concepts from the OCL specification [51, A.1]. Set theory comprises very basic concepts describing structures, only admitting the subsumption of elements in sets and set hierarchies. It holds no further information about the semantics of the elements.

Also, the FDMM formalism introduced by Fill et al. [24] addressing conceptual modeling domains in a wider variety uses set theory to specify metamodels and models. The authors explicitly aim at a formalization of metamodels realized with the metamodeling platform ADOxx [5] and do not claim to be applicable for platform-independent specifications.

Neither graph theory, basis of KM3, nor set theory, basis of FDMM and the MOF formalization by Burger, do justice to the linguistic character of modeling languages and provide canonical concepts for the definition of a set of terms and for instantiation, an essential characteristic of modeling languages. Therefore the technique and semantics of this conformance of a model to its metamodel has to be constructed ad-hoc and lacks the beneficial knowledge stack of established theories dealing with linguistic structures.

2.5 Formalisms based on logic

Formal languages as defined in mathematical logic inherently comprise the concept of instantiation as interpretation of the signature in logical terms, and they provide a rich knowledge base about their properties. Therefore, in current research, the notion of modeling languages as formal languages in the sense of mathematical logic is receiving increasing attention [18,28,53,56,63].

In their investigation of formal foundations of domain-specific languages, Jackson and Sztipanovits [33] introduce term algebras to handle models . They indeed treat modeling languages as formal languages with a signature and an alphabet. Nevertheless, they mostly abandon the notion of conceptual modeling in the formalism. A model is defined as a set of terms without explicating the equivalents of its constituents, i.e., objects, relations, and attributes. Without a procedure to define object and relation types, the approach lacks the instantiation relation between elements in the metamodel and the model that is characteristic for modeling languages.

Telos [40] builds on the premise that the concepts of entities and relations are omitted and replaced by propositions constituting the knowledge base. The choice of typed first-order logic for the formalization of these propositions is natural and explained in great detail in [41]. Knowledge in Telos is represented solely as a set of sentences in the formal language. In our approach, on the other hand, we do not adopt the transformation of models into propositions but rather directly deal with the ubiquitous concepts of objects and relations and an instantiation hierarchy between models and metamodels. This leads to a different view on models. In Telos, a model is constituted by FOL statements, whereas in our approach these FOL statements are used as constraints restricting valid expressions using the proposed signature.

In his work on the theory of conceptual models, Thalheim [63] describes modeling languages as being based on a signature Σ comprising a set of postulates, i.e., sentences expressed with elements of Σ. Models are defined as language structures satisfying the postulates, which canonically corresponds to the concept of instantiation of a metamodel. We go one step further and concretely point out how to capture the core concepts of a modeling language in a signature Σ to unify the method of formalizing a language. This then enables us to investigate the class of formal modeling languages, compare formalized languages, reuse components and develop generic methods for language fusion, model transformation, etc., independent of a concrete language.

In summary, the literature review suggests that the structure of modeling languages, including their linguistic character, can be grounded in the concepts of formal languages. Therefore, in the work at hand, we propose a formal definition of modeling languages in which we concretely specify the modeling concepts and their formal equivalent in logical terms with the prospect of successive elaboration.

We adopt the four-layer metamodeling stack architecture that is widely used in practice nowadays [1,11]. Even though it is often seen as insufficient for those modeling situations that motivated the multi-level modeling approach [26], we agree with the argument in [47] that a language offering means to model an intension/extension relation (extension denominating the elements characterized by their intension

concept, akin to the *instantiation* relation [15]) allows to model elements of different levels. Such a relation type tackles some of the core characteristics of multi-level modeling. A possibility of realizing this intension/extension relation is the employment of the power type pattern. We will use this idea in Sect. 5 to demonstrate how easily the MetaMorph approach can be extended to encompass additional, advanced tools in conceptual modeling.

3 The MetaMorph formalism: defining formal modeling languages

The intended definition shall serve as a cornerstone for a common way of formalizing modeling languages, thereby becoming comparable, reusable, and modularizable. A formal definition for modeling languages in general enables an investigation of common features of the resulting subclass of formal languages as well as a sound mathematical foundation for their functionality. We build on a survey conducted by Kern et al. [38] on common concepts in the meta^{2}models of six established metamodeling platforms. The definition below incorporates all concepts identified in at least half of the investigated platforms. These are object types, relation types (binary), attributes (multi-value), inheritance (for object types), and a constraint language. In accordance with the state of the art in conceptual modeling research, we replaced the term *inheritance* used in [38] with *specialization*, a more accurate notion for this dependency. Other concepts identified in [38] which are not yet included in MetaMorph are roles, ports, specialization of relations, n-ary relations, and models in the sense of model types.

These concepts mainly coincide with the core concepts introduced for conceptual modeling of information systems by Olivé [53]. Additional concepts mentioned in Olivé's work which are of high interest but not yet included in our approach are derived types and generic relation types. The concept of power types also mentioned in [53] will be used to demonstrate the ease of extending the MetaMorph formalism in Sect. 5.

3.1 A definition based on predicate logic

The MetaMorph formalism builds on typed (also called sorted) predicate logic. The mathematical basics can be found in textbooks on logic or mathematics for computer science, e.g., [23,48]. Some remarks on notation: To ease the differentiation between language and model-level, we use capital letters for the symbols of the language and lowercase letters for the elements of the model.

Definition 1 A (formal) modeling language $\mathcal{L}$ consists of a typed signature $\Sigma = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}$ and a set $\mathbf{C}$ of sentences in $\mathcal{L}$ for the constraints, where:

- $\mathcal{S}$ is a set of types, which can be further divided into three disjoint subsets $\mathcal{S}_O$, $\mathcal{S}_R$, and $\mathcal{S}_D$ for object types, relation types and data types;
  - the type set $\mathcal{S}_O$ is strictly partially ordered with order relation $<_O \subseteq \mathcal{S}_O \times \mathcal{S}_O$ to indicate the specialization relation between the corresponding object types;
  - the type set $\mathcal{S}_D$ can contain simple types $\mathbf{T}$ for value domains of single-value attributes, or product types $\mathbf{T}' = \mathbf{T}_1 \times \mathbf{T}_2 \times \cdots \times \mathbf{T}_n$ and unions thereof for value domains of n-ary multi-value attributes ($n > 1$), where the ith value is of type $\mathbf{T}_i \in \mathcal{S}_D \cup \mathcal{S}_O \cup \mathcal{S}_R$;
- $\mathcal{F}$ is a set of typed function symbols such that:
  - for each relation type $\mathbf{R}$ in $\mathcal{S}_R$ there exist two function symbols $F_s^{\mathbf{R}}$ and $F_t^{\mathbf{R}}$ with domain type $\mathbf{R} \in \mathcal{S}_R$ and codomain type $\mathbf{O}_s, \mathbf{O}_t \in \mathcal{S}_O$ assigning the source and target object types to a relation;
  - for each single-value attribute $\mathbf{A}$ of an object or relation type $\mathbf{T}$ there exists a function symbol $F^{\mathbf{A}}$ with domain type $\mathbf{T}$ and codomain type a simple type in $\mathcal{S}_D$ or an element in $\mathcal{S}_O$ or $\mathcal{S}_R$ assigning the simple data type or referenced object type or relation type to the attribute;
  - for each multi-value attribute $\mathbf{A}$ of an object or relation type $\mathbf{T}$ there exists a function symbol $F^{\mathbf{A}}$ with domain type $\mathbf{T}$ and codomain type a product type in $\mathcal{S}_D$ or unions thereof;
- $\mathcal{R}$ is a set of typed relation symbols;
- $\mathcal{C}$ is a set of typed constants to specify the possible values c_i of a simple type $\mathbf{T} \in \mathcal{S}_D$ of the attributes;
- the set $\mathbf{C}$ is a set of sentences in $\mathcal{L}$ constraining the possible models, also called the postulates of the language.

This definition explicates the formalization of the essential modeling concepts of a language, i.e., object types and specialization, binary directed relation types and single- or multi-value attributes. Note that the definition does not prohibit the existence of additional symbols in the signature, so broader concepts like n-ary relation types can optionally be included and are topic of further investigation. Also, structures beyond the visual elements of a model can be included, e.g., paths as transitive relations or substructures comprising several elements.

We want to point out that relation types are defined on the same level as object types, not subordinate to them. This highlights their significance for a model beyond mere arrows and allows for defining attributes of relations, multiple rela-

tions of the same type between the same two objects, as well as for specialization of relation types.

Specialization of object types $\mathbf{O}_1 <_O \mathbf{O}_2$ means, that all elements of the subtype $\mathbf{O}_1$ are indeed elements of the supertype $\mathbf{O}_2$. Therefore, the elements of $\mathbf{O}_1$ also belong to the domain of an attribute function $F^{\mathbf{A}} : \mathbf{O}_2 \rightarrow \mathbf{T}$ for an attribute $\mathbf{A}$ possessed by $\mathbf{O}_2$. This implies that elements of type $\mathbf{O}_1$ also possess this attribute and that attributes inherited from the supertype cannot be altered for the subtype. Furthermore, the specialization relation $<_O$ is defined as a strict partial order and therefore allows for multiple supertypes for a subtype.

With the data types and constants, we can define attribute domains like integers via specifying a type called $\mathbb{N}$ and constant symbols 1,2,3, …in $\mathcal{C}$ of type $\mathbb{N}$ for the numbers, or enumeration lists like a person's gender via specifying a type called *gender* and constant symbols *male, female*, and *else* in $\mathcal{C}$. The elements of the simple or product types of $\mathcal{S}_D$ are typically not visible in graphical models. They are exclusively used for specifying attribute domains.

Note that if we assume the set of constants for attribute domains to be finite, models are always finite because by construction they contain only finitely many objects and relations.

Definition 2 A model $\mathcal{M}$ of a language $\mathcal{L}$ with typed signature $\Sigma = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}$ is an $\mathcal{L}$-structure conforming to the language constraints $\mathbf{C}$, i.e., $\mathcal{M}$ consists of

- a universe $\mathcal{U}$ of typed elements respecting the type hierarchy, that is
  - for each $\mathbf{T}$ in $\mathcal{S}$ there exists a set $\mathcal{U}_{\mathbf{T}} \subseteq \mathcal{U}$ and $\mathcal{U} = \bigcup_{\mathbf{T} \in \mathcal{S}} \mathcal{U}_{\mathbf{T}}$;
  - all sets $\mathcal{U}_{\mathbf{T}}$ for $\mathbf{T} \in \mathcal{S}_O \cup \mathcal{S}_R$ have to be pairwise disjoint except for sets $\mathcal{U}_{\mathbf{O}_1}$ and $\mathcal{U}_{\mathbf{O}_2}$ with $\mathbf{O}_1, \mathbf{O}_2 \in \mathcal{S}_O$ where $\mathbf{O}_1 <_O \mathbf{O}_2$. In this case $\mathcal{U}_{\mathbf{O}_1}$ must be a subset of $\mathcal{U}_{\mathbf{O}_2}$, i.e., $\mathcal{U}_{\mathbf{O}_1} \subseteq \mathcal{U}_{\mathbf{O}_2}$;
  - all sets $\mathcal{U}_{\mathbf{T}}$ with $\mathbf{T} = \mathbf{T}_1 \times \mathbf{T}_2 \times \cdots \times \mathbf{T}_n$ a product type in $\mathcal{S}_D$ consist of tuples $(x_1, x_2, \ldots x_n) \in \mathcal{U}_{\mathbf{T}_1} \times \mathcal{U}_{\mathbf{T}_2} \times \cdots \times \mathcal{U}_{\mathbf{T}_n}$;
- an interpretation of the function symbols in $\mathcal{L}$, i.e., for each function symbol $F \in \mathcal{F}$ with domain type $\mathbf{T}_1 \times \ldots \times \mathbf{T}_n$ and codomain type $\mathbf{T}$ a function $f : \mathcal{U}_{\mathbf{T}_1} \times \ldots \times \mathcal{U}_{\mathbf{T}_n} \rightarrow \mathcal{U}_{\mathbf{T}}$;
- an interpretation of the relation symbols in $\mathcal{L}$, i.e., for each relation symbol $R \in \mathcal{R}$ with domain type $\mathbf{T}_1 \times \ldots \times \mathbf{T}_m$ a relation $r \subseteq \mathcal{U}_{\mathbf{T}_1} \times \ldots \times \mathcal{U}_{\mathbf{T}_m}$;
- for each simple type $\mathbf{T} \in \mathcal{S}_D$ and constant $C \in \mathcal{C}$ of type $\mathbf{T}$ an interpretation $c \in \mathcal{U}_{\mathbf{T}}$;
- for each constraint ϕ in $\mathbf{C}$ the model $\mathcal{M}$ satisfies ϕ, i.e., $\mathcal{M} \models \phi$.

Fig. 2 Notation excerpt of the CoChaCo method [37]

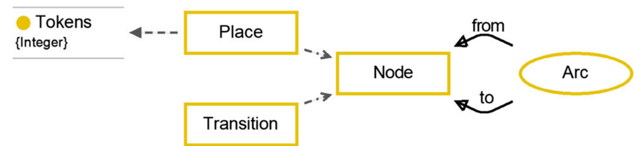

Fig. 3 A metamodel of Petri Nets

This definition of models as language structures goes beyond a visualization and considers models as knowledge structures as described in [12]. Thereby, we overcome several shortcomings of graphical representations, like the missing depiction of attributes and their domains in models or the visual mixing of the metarelation specialization with the definition of relation types in metamodels.

3.2 Running example Petri Nets

We will now illustrate the definition on the example of the Petri Net modeling language. For the visualization of the metamodel, we use the notation of CoChaCo, a method to support the creative process of modeling method design [37]. This method comprises concrete syntax for most of the concepts listed in Definition 1 with a slightly different naming, see Fig. 2.

Example 1 The Petri Net modeling language $\mathcal{PN}$
The Petri Net metamodel depicted in Fig. 3 comprises three object types **Node** (**No**), **Place** (**Pl**), and **Transition** (**Tr**) constituting $\mathcal{S}_O$. Thereby **Place** and **Transition** specialize **Node**, i.e., **Place** $<_O$ **Node** and **Transition** $<_O$ **Node**. Furthermore, the language comprises only one relation type **Arc** element of $\mathcal{S}_R$. For the attribute **Tokens** of object type **Place**, we need a type $\mathbb{N}_0$ with the usual addition $+_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$ and order relation $<_{\mathbb{N}_0} \subseteq \mathbb{N}_0 \times \mathbb{N}_0$, i.e., the set of all tuples (x, y) in $\mathbb{N}_0 \times \mathbb{N}_0$ where $x < y$. Additionally, we specify constants in $\mathcal{C} = \{0, 1, 2, \ldots\}$ all of type $\mathbb{N}_0$. The set $\mathcal{S}$ of types is then the union $\mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D$={**Node**, **Place**, **Transition**, **Arc**, $\mathbb{N}_0$}. For the relation type **Arc**, we have to specify the source and target object types by introducing two function symbols $F_s^{\mathbf{Arc}}$ and $F_t^{\mathbf{Arc}}$ both with domain **Arc** and codomain **Node**. For the attribute **Tokens**, we introduce a function symbol $F^{\mathbf{Tokens}}$ with domain **Place** and codomain $\mathbb{N}_0$ assigning each place instance a number of tokens.

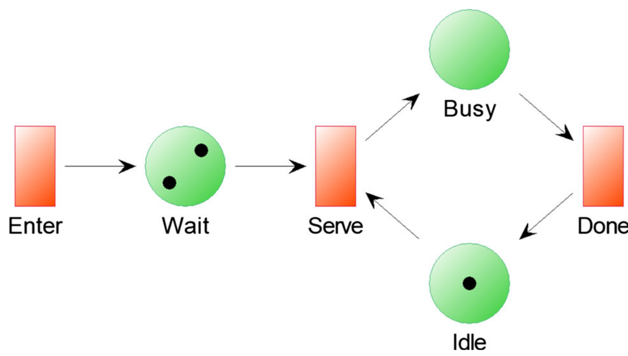

Fig. 4 A Petri Net model depicting a simple barber shop scenario

The signature of $\mathcal{PN}$ looks as follows:

$$\Sigma = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\},\ \mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D \tag{1}$$

$$\mathcal{S}_O = \{\textbf{Node}, \textbf{Place}, \textbf{Transition}\}, \tag{2}$$

$$\textbf{Place} <_O \textbf{Node},\ \textbf{Transition} <_O \textbf{Node} \tag{3}$$

$$\mathcal{S}_R = \{\textbf{Arc}\}, \tag{4}$$

$$\mathcal{S}_D = \{\mathbb{N}_0\}, \tag{5}$$

$$\mathcal{F} = \{F_s^{\textbf{Arc}} : \textbf{Arc} \to \textbf{Node},\ F_t^{\textbf{Arc}} : \textbf{Arc} \to \textbf{Node}, F^{\textbf{Tokens}} : \textbf{Place} \to \mathbb{N}_0, +_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0\}, \tag{6}$$

$$\mathcal{R} = \{<_{\mathbb{N}_0} \subseteq \mathbb{N}_0 \times \mathbb{N}_0\}, \tag{7}$$

$$\mathcal{C} = \{0, 1, 2, \ldots\},\ \text{all of type } \mathbb{N}_0. \tag{8}$$

Finally, we have to define the constraints of the language. These rules are not contained in a graphical metamodel. In existing specifications, they are mainly specified with natural language or OCL. In the predicative formalization of MetaMorph constraints are an integral part of the language. Following four sentences written in the alphabet of $\mathcal{PN}$ ensure **Node** to be abstract, i.e., any element in **Node** lies either in **Place** or in **Transition** (9), as well as the alternation of types of the elements connected by an arc (10, 11) and the prohibition of multiple arcs between the same two elements (12). For ease of readability, we abuse the notation $\forall x \in \textbf{T}$ for x being of type $\textbf{T}$ instead of using the type specific quantifier $\forall_{\textbf{T}} x$.

$$\forall x \in \textbf{No}\ \exists y \in \textbf{Pl}, z \in \textbf{Tr}\ (x = y \vee x = z) \tag{9}$$

$$\nexists x, y \in \textbf{Pl}, u \in \textbf{Arc}\ (F_s^{\textbf{Arc}}(u) = x \wedge F_t^{\textbf{Arc}}(u) = y) \tag{10}$$

$$\nexists x, y \in \textbf{Tr}, u \in \textbf{Arc}\ (F_s^{\textbf{Arc}}(u) = x \wedge F_t^{\textbf{Arc}}(u) = y) \tag{11}$$

$$\forall u, v \in \textbf{Arc}\ ((F_s^{\textbf{Arc}}(u) = F_s^{\textbf{Arc}}(v) \wedge F_t^{\textbf{Arc}}(u) = F_t^{\textbf{Arc}}(v)) \implies u = v) \tag{12}$$

Example 2 A Petri Net model

A Petri Net model depicting a simple barber shop scenario is shown in Fig. 4. Its formalization, i.e., the corresponding $\mathcal{PN}$-structure, looks as follows: The universe of places $\mathcal{U}_P$ contains three elements $\mathcal{U}_P = \{\textbf{w}(ait), \textbf{b}(usy), \textbf{i}(dle)\}$. The universe of transitions $\mathcal{U}_T$ comprises three elements $\mathcal{U}_T = \{\textbf{e}(nter), \textbf{s}(erve), \textbf{d}(one)\}$. Six arc elements exist in $\mathcal{U}_A = \{\textbf{a}_1, \textbf{a}_2, \textbf{a}_3, \textbf{a}_4, \textbf{a}_5, \textbf{a}_6\}$ with source and target $f_s^{\textbf{Arc}}(\textbf{a}_1) = \textbf{e}$, $f_t^{\textbf{Arc}}(\textbf{a}_1) = \textbf{w}$, $f_s^{\textbf{Arc}}(\textbf{a}_2) = \textbf{w}$, $f_t^{\textbf{Arc}}(\textbf{a}_2) = \textbf{s}$, $f_s^{\textbf{Arc}}(\textbf{a}_3) = \textbf{s}$, $f_t^{\textbf{Arc}}(\textbf{a}_3) = \textbf{b}$, $f_s^{\textbf{Arc}}(\textbf{a}_4) = \textbf{b}$, $f_t^{\textbf{Arc}}(\textbf{a}_4) = \textbf{d}$, $f_s^{\textbf{Arc}}(\textbf{a}_5) = \textbf{d}$, $f_t^{\textbf{Arc}}(\textbf{a}_5) = \textbf{i}$, $f_s^{\textbf{Arc}}(\textbf{a}_6) = \textbf{i}$, and $f_t^{\textbf{Arc}}(\textbf{a}_6) = \textbf{s}$. For the attribute type and values, the natural numbers $\mathbb{N}_0$ are included in the model, $\mathcal{U}_{\mathbb{N}_0} = \{0, 1, 2, \ldots\}$. The instantiation of the attribute *Tokens* looks as follows: $f^{\textbf{Tokens}}(\textbf{w}) = 2$, $f^{\textbf{Tokens}}(\textbf{b}) = 0$ and $f^{\textbf{Tokens}}(\textbf{i}) = 1$. The interpretations of the function and relation symbols $+_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ and $<_{\mathbb{N}_0} \subseteq \mathbb{N}_0 \times \mathbb{N}_0$ are defined according to the usual addition and order relation on natural numbers. We can easily check that the formalized model satisfies all postulates (9)–(12) of the language $\mathcal{PN}$.

Notice that the formalized model representation in Example 2 and the graphical model representation in Fig. 4 show the same thing. They are merely alternative ways of describing a system but with different merits. Whereas the graphical model representation is easy and fast to comprehend, the formal model representation is precise and complete, as attribute values are often not legible from a pictorial model. This can be compared to the different representation forms of a graph—once as a graphical depiction and once as an adjacent matrix.

4 `M2FOL`: metamodel 2 first-order logic a formal modeling language for metamodels

Metamodels are models themselves expressed in a metamodeling language. We propose a formal modeling language in the sense of Definition 1 for metamodels called `M2FOL`, i.e., a metamodeling language to be exact. This language is capable of describing precisely the concepts explicated in Definition 1. In general, meta^{2}models of metamodeling languages are supposed to be self-describing, which results in a four-layer metamodeling stack as depicted in Fig. 5. We will show that the metamodel of `M2FOL`, a meta^{2}model by nature, also partakes of this property.

4.1 Definition of `M2FOL`

We stick to the notational convention of capital letters for elements on the language-level and lowercase letters for elements on the model-level. To indicate the metalevel of

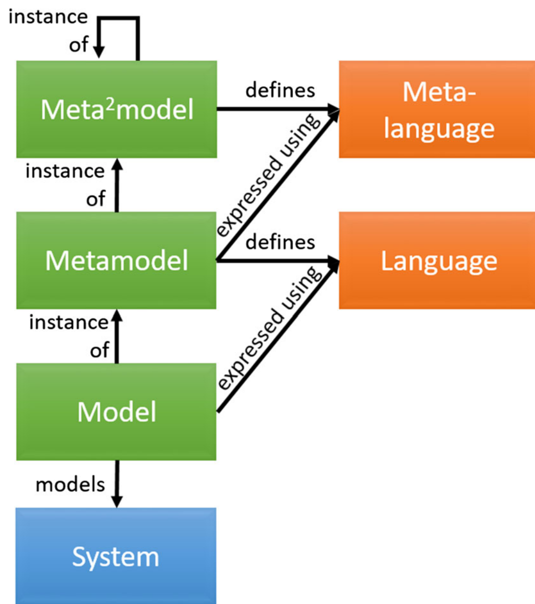

Fig. 5 The four-layer metamodeling stack based on [43]

M2FOL and metamodels, we use the typewriter font for meta symbols and elements. For ease of readability, we write $F : X \to Y$ when F is a function with domain type X and codomain type Y. Nevertheless, the instantiation is then a function $f : \mathcal{U}_X \to \mathcal{U}_Y$ defined on universes of typed elements. To be consistent in the naming of the symbols in M2FOL we distinguish between an attribute type on meta-level and an attribute as the concrete assignment of a value to an element on model-level.

With M2FOL, we want to model **object types** and *specialization* relations between them, **relation types** connected to their *from* and *to* object types, **attribute types** and their **data types**, and possible **data**. According to Definition 1, all the bold concepts constitute a type in $\mathcal{S}_O$ in M2FOL, whereas all italic concepts make up a type in $\mathcal{S}_R$ in M2FOL. The types *specialization*, *from*, and *to* furthermore require assignment functions for source and target specification. Data types and data are necessary for defining attribute domains and their values, e.g., the domain $\mathbb{N}_{0...10}$ and values $\{0, 1, 2, \ldots, 9, 10\}$ or an enumeration list domain *gender* with values *male*, *female*, *else*. Attribute types need the assignment of owning type and value domain.

Definition 3 The metalanguage M2FOL is a modeling language with signature $\Sigma = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}$ with the set of types split in $\mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D$, where:

- $\mathcal{S}_O$ consists of the types O(bject) T(ype), R(elation) T(ype), A(ttribute) T(ype), D(ata) T(ype), and D(ata), furthermore two supertypes: ORT(ype), and DORT(ype): $\mathcal{S}_O = \{\mathtt{OT}, \mathtt{RT}, \mathtt{AT}, \mathtt{DT}, \mathtt{D}, \mathtt{ORT}, \mathtt{DORT}\}$;
  - The types OT, and RT, specialize ORT, the types ORT, and DT specialize DORT: $\mathtt{OT} <_O \mathtt{ORT}$, $\mathtt{RT} <_O \mathtt{ORT}$, $\mathtt{ORT} <_O \mathtt{DORT}$, $\mathtt{DT} <_O \mathtt{DORT}$;
- $\mathcal{S}_R$ consists of the types Spec(ialization), Fr(om), and To: $\mathcal{S}_R = \{\mathtt{Spec}, \mathtt{Fr}, \mathtt{To}\}$;
- $\mathcal{S}_D$ contains product types $\mathtt{DORT}^n$ for all $n > 1$ as well as a type $\mathtt{T}_{\mathtt{DORT}}$ for the union of all $\mathtt{DORT}^n$: $\mathtt{T}_{\mathtt{DORT}} = \bigcup_i \mathtt{DORT}^i$;
- the set of function symbols consists of following elements:
  - two symbols $\mathtt{F}_s^{\mathtt{Spec}}$ and $\mathtt{F}_t^{\mathtt{Spec}}$ assigning source and target to Spec-typed relations: $\mathtt{F}_s^{\mathtt{Spec}} : \mathtt{Spec} \to \mathtt{OT}$, $\mathtt{F}_t^{\mathtt{Spec}} : \mathtt{Spec} \to \mathtt{OT}$;
  - two symbols $\mathtt{F}_s^{\mathtt{Fr}}$ and $\mathtt{F}_t^{\mathtt{Fr}}$ assigning source and target to Fr-typed relations: $\mathtt{F}_s^{\mathtt{Fr}} : \mathtt{Fr} \to \mathtt{RT}$, $\mathtt{F}_t^{\mathtt{Fr}} : \mathtt{Fr} \to \mathtt{OT}$;
  - two symbols $\mathtt{F}_s^{\mathtt{To}}$ and $\mathtt{F}_t^{\mathtt{To}}$ assigning source and target to To-typed relations: $\mathtt{F}_s^{\mathtt{To}} : \mathtt{To} \to \mathtt{RT}$, $\mathtt{F}_t^{\mathtt{To}} : \mathtt{To} \to \mathtt{OT}$;
  - two symbols $\mathtt{F}_{val}$ and $\mathtt{F}_{type}$ assigning to an attribute type its value domain and the object or relation type it belongs to. The value assignment can be a reference or a n-valued type in $\mathtt{DORT}^n$: $\mathtt{F}_{val} : \mathtt{AT} \to \bigcup_i \mathtt{DORT}^i$, $\mathtt{F}_{type} : \mathtt{AT} \to \mathtt{ORT}$;
  - a symbol $\mathtt{F}_{\mathtt{DT}}$ to assign a data type to a data element: $\mathtt{F}_{\mathtt{DT}} : \mathtt{D} \to \mathtt{DT}$;
- $\mathcal{R}$ consists of a symbol $<_{\mathtt{OT}}$ transitively extending the specialization relation given by Spec to a strict partial order on the set of object types $\mathcal{R} = \{<_{\mathtt{OT}} \subseteq \mathtt{OT} \times \mathtt{OT}\}$.

We formulate the postulates of the language. For brevity, we use the abbreviation xry for relation r of type $\mathbf{T}$, x being $\mathtt{F}_s^{\mathbf{T}}(r)$ and y being $\mathtt{F}_t^{\mathbf{T}}(r)$.

The following constraints ensure $<_{\mathtt{OT}}$ being the transitive closure of Spec under the assumption that all universes are finite:

$$\forall x, y \in \mathtt{OT}, u \in \mathtt{Spec}\ (xuy \implies x <_{\mathtt{OT}} y) \quad (13)$$

$$\forall x, y \in \mathtt{OT}\ \exists z \in \mathtt{OT}, u \in \mathtt{Spec}\ (x <_{\mathtt{OT}} y \implies xuy \lor (xuz \land z <_{\mathtt{OT}} y)) \quad (14)$$

We furthermore require $<_{\mathtt{OT}}$ to be a strict partial order, i.e., $<_{\mathtt{OT}}$ is transitive, irreflexive and antisymmetric. The absence of cyclic specialization and self-specialization follows from the properties of $<_{\mathtt{OT}}$.

The following constraints ensure the existence and uniqueness of To and Fr objects of a relation (15–17), and the abstractness of the types ORT and DORT (18–19):

$$\forall x \in \mathtt{RT}\ \exists y, z \in \mathtt{OT}, u \in \mathtt{Fr}, v \in \mathtt{To}\ (xuy \land xvz) \quad (15)$$

$$\nexists u, v \in \mathtt{Fr}\ (\mathtt{F}_s^{\mathtt{Fr}}(u) = \mathtt{F}_s^{\mathtt{Fr}}(v) \land u \neq v) \quad (16)$$

Table 1 Algorithm to deduce a formal modeling language signature from its M2FOL metamodel specification

	M2FOL (meta)model to language-signature	Mapping	Application to the Petri Net metamodel
1.	Each metamodel element o in the set $\mathcal{U}_{\mathrm{OT}}$ defines an object type **O** of the language. The specialization relation $<_{\mathrm{OT}} \subseteq \mathcal{U}_{\mathrm{OT}} \times \mathcal{U}_{\mathrm{OT}}$ must be adopted to the types	$\mathtt{o} \in \mathcal{U}_{\mathrm{OT}} \rightarrowtail \mathbf{O} \in \mathcal{S}_O$, $<_{\mathrm{OT}} \rightarrowtail <_O$	$\mathtt{node} \in \mathcal{U}_{\mathrm{OT}} \rightarrowtail \mathbf{Node} \in \mathcal{S}_O$, $\mathtt{place} \in \mathcal{U}_{\mathrm{OT}} \rightarrowtail \mathbf{Place} \in \mathcal{S}_O$, $\mathtt{trans} \in \mathcal{U}_{\mathrm{OT}} \rightarrowtail \mathbf{Trans} \in \mathcal{S}_O$
2.	Each metamodel element r in the set $\mathcal{U}_{\mathrm{RT}}$ defines a relation type **R** of the language	$\mathtt{r} \in \mathcal{U}_{\mathrm{RT}} \rightarrowtail \mathbf{R} \in \mathcal{S}_R$	$\mathtt{arc} \in \mathcal{U}_{\mathrm{RT}} \rightarrowtail \mathbf{Arc} \in \mathcal{S}_R$
3.	For each relation type $\mathtt{r} \in \mathcal{U}_{\mathrm{RT}}$, there exist an element s of type From and an element t of type To and both relation elements have as source element r, $\mathtt{f}_s^{\mathtt{Fr}}(\mathtt{s}) = \mathtt{r}$, $\mathtt{f}_s^{\mathtt{Fr}}(\mathtt{t}) = \mathtt{r}$. The assignment $\mathtt{f}_t^{\mathtt{To}}(\mathtt{s}) = \mathtt{o}_s$ indicates the source object type of **R**, $\mathtt{f}_t^{\mathtt{To}}(\mathtt{t}) = \mathtt{o}_t$ indicates the target object type of **R**	$\mathtt{r}, \mathtt{s}, \mathtt{o}_s \rightarrowtail \mathtt{F}_s^{\mathrm{R}} : \mathbf{R} \to \mathbf{O}_s$; $\mathtt{r}, \mathtt{t}, \mathtt{o}_t \rightarrowtail \mathtt{F}_t^{\mathrm{R}} : \mathbf{R} \to \mathbf{O}_t$	$\mathtt{arc}, \mathtt{f}_t^{\mathtt{Fr}}(\mathtt{a_from}) = \mathtt{n} \rightarrowtail \mathtt{F}_s^{\mathrm{Arc}} : \mathbf{Arc} \to \mathbf{Node}$; $\mathtt{arc}, \mathtt{f}_t^{\mathtt{To}}(\mathtt{a_to}) = \mathtt{n} \rightarrowtail \mathtt{F}_t^{\mathrm{Arc}} : \mathbf{Arc} \to \mathbf{Node}$
4.	Each metamodel element dt in $\mathcal{U}_{\mathrm{DT}}$ defines a data type **DT** of the language. Each metamodel element d in $\mathcal{U}_{\mathrm{D}}$ with $\mathtt{f}_{\mathrm{DT}}(\mathtt{d}) = \mathtt{dt}$ becomes a constant symbol C_d in $\mathcal{C}$ of type **DT**	$\mathtt{dt} \in \mathcal{U}_{\mathrm{DT}} \rightarrowtail \mathbf{DT} \in \mathcal{S}_D$; $\mathtt{d} \in \mathcal{U}_{\mathrm{D}} \rightarrowtail C_{\mathtt{d}} \in \mathcal{C}$	$\mathbb{N} \in \mathcal{U}_{\mathrm{DT}} \rightarrowtail \mathbb{N} \in \mathcal{S}_{DT}$; $\mathtt{i} \in \mathcal{U}_{\mathrm{D}}$, $\mathtt{f}_{DT}(i) = \mathbb{N} \rightarrowtail \mathtt{i} \in \mathcal{C}$ of type $\mathbb{N}$
5.	Each metamodel element a in the set $\mathcal{U}_{\mathrm{AT}}$ defines a function symbol $\mathtt{F}^{\mathtt{a}}$ of the language. The object or relation type that a belongs to, i.e., the domain of $\mathtt{F}^{\mathtt{a}}$, is given by the assignment $\mathtt{f}_{type}(\mathtt{a}) = \mathtt{t}_{ty} \in \mathcal{U}_{\mathrm{OT}} \cup \mathcal{U}_{\mathrm{RT}}$, its value range, i.e., codomain, by $\mathtt{f}_{val}(\mathtt{a}) = (\mathtt{t}_{v_1}, \ldots, \mathtt{t}_{v_n}) \in (\mathcal{U}_{\mathrm{OT}} \cup \mathcal{U}_{\mathrm{RT}} \cup \mathcal{U}_{\mathrm{DT}})^n$	$\mathtt{a}, \mathtt{t}_{ty}, \mathtt{t}_{\bar{v}} \rightarrowtail F^{\mathtt{a}} : T_{ty} \to T_{\bar{v}}$	$\mathtt{tok}, \mathtt{f}_{type}(\mathtt{tok}) = \mathtt{place}$, $\mathtt{f}_{val}(\mathtt{tok}) = \mathbb{N} \rightarrowtail F^{Tokens} : \mathbf{Place} \to \mathbb{N}$
6.	The constraints of the language have to be added manually, because this information is not determined by the metamodel		

$$\nexists u, v \in \mathtt{To}\ (\mathtt{F}_s^{\mathtt{To}}(u) = \mathtt{F}_s^{\mathtt{To}}(v) \wedge u \neq v) \quad (17)$$

$$\forall x \in \mathtt{ORT}\ \exists y \in \mathtt{OT}, z \in \mathtt{RT}(x = y \vee x = z) \quad (18)$$

$$\forall x \in \mathtt{DORT}\ \exists y \in \mathtt{ORT}, z \in \mathtt{DT}(x = y \vee x = z) \quad (19)$$

4.2 Running example Petri Nets

With this language, we now can transfer the graphical metamodel of Fig. 3 to a formal M2FOL-model.

Example 3 The Petri Net metamodel $\mathtt{M}_{PN}$
The universe of object types $\mathcal{U}_{\mathrm{OT}}$ comprises three elements: n(ode), p(lace), and tr(ansition). The universe of relation types $\mathcal{U}_{\mathrm{RT}}$ contains one element a(rc). One element tok(ens) is contained in the universe of attribute types $\mathcal{U}_{\mathrm{AT}}$. The universe $\mathcal{U}_{\mathtt{Spec}}$ contains the specialization relations p_n between p and n as well as tr_n between tr and n. $\mathcal{U}_{\mathtt{Fr}}$ contains the relation a_from of the source element assignment to the relation type a. $\mathcal{U}_{\mathtt{To}}$ contains the relation a_to of the target element assignment to the relation type a. For these four elements, the corresponding source and target elements have to be assigned: $\mathtt{f}_s^{\mathtt{Spec}}(\mathtt{p_n}) = \mathtt{p}$, $\mathtt{f}_t^{\mathtt{Spec}}(\mathtt{p_n}) = \mathtt{n}$, $\mathtt{f}_s^{\mathtt{Spec}}(\mathtt{tr_n}) = \mathtt{tr}$, $\mathtt{f}_t^{\mathtt{Spec}}(\mathtt{tr_n}) = \mathtt{n}$, $\mathtt{f}_s^{\mathtt{Fr}}(\mathtt{a_from}) = \mathtt{a}$, $\mathtt{f}_t^{\mathtt{Fr}}(\mathtt{a_from}) = \mathtt{n}$, $\mathtt{f}_s^{\mathtt{To}}(\mathtt{a_to}) = \mathtt{a}$, $\mathtt{f}_t^{\mathtt{To}}(\mathtt{a_to}) = \mathtt{n}$. From Spec the transitive order relation $<_{\mathrm{OT}}$ is deduced: $<_{\mathrm{OT}} = \{(\mathtt{p}, \mathtt{n}), (\mathtt{tr}, \mathtt{n})\}$. Furthermore, there are data values $\{0, 1, 2, \ldots\}$ in $\mathcal{U}_{\mathrm{D}}$ all of type $\mathbb{N}_0 \in \mathcal{U}_{\mathrm{DT}}$, $\mathtt{f}_{DT}(i) = \mathbb{N}_0\ \forall i \in \mathcal{U}_{\mathrm{D}}$. These are needed for the value domain of the attribute type tok, an attribute assigned to p: $\mathtt{f}_{type}(\mathtt{tok}) = \mathtt{p}$, $\mathtt{f}_{val}(\mathtt{tok}) = \mathbb{N}_0$. In short, this can be written as follows:

$$\mathcal{U}_{\mathrm{OT}} = \{\mathtt{n}(ode),\ \mathtt{p}(lace),\ \mathtt{tr}(ansition)\}, \quad (20)$$

$$\mathcal{U}_{\mathrm{RT}} = \{\mathtt{a}(rc)\},\ \mathcal{U}_{\mathrm{AT}} = \{\mathtt{tok}(en)\}, \quad (21)$$

$$\mathcal{U}_{\mathtt{Spec}} = \{\mathtt{p_n}, \mathtt{tr_n}\}, \quad (22)$$

$$\mathcal{U}_{\mathtt{Fr}} = \{\mathtt{a_from}\},\ \mathcal{U}_{\mathtt{To}} = \{\mathtt{a_to}\}, \quad (23)$$

$$\mathcal{U}_{\mathrm{DT}} = \{\mathbb{N}_0\},\ \mathcal{U}_{\mathrm{D}} = \{0, 1, 2, \ldots\}, \quad (24)$$

$$\mathcal{U}_{\mathrm{ORT}} = \{\mathtt{n}, \mathtt{p}, \mathtt{tr}, \mathtt{a}\}, \mathcal{U}_{\mathrm{DORT}} = \{\mathtt{n}, \mathtt{p}, \mathtt{tr}, \mathtt{a}, \mathbb{N}_0\} \quad (25)$$

$$<_{\mathrm{OT}} = \{(\mathtt{p}, \mathtt{n}), (\mathtt{tr}, \mathtt{n})\} \quad (26)$$

$$\mathtt{f}_s^{\mathtt{Spec}}(\mathtt{p_n}) = \mathtt{p},\ \mathtt{f}_t^{\mathtt{Spec}}(\mathtt{p_n}) = \mathtt{n}, \quad (27)$$

$$\mathtt{f}_s^{\mathtt{Spec}}(\mathtt{tr_n}) = \mathtt{tr},\ \mathtt{f}_t^{\mathtt{Spec}}(\mathtt{tr_n}) = \mathtt{n}, \quad (28)$$

$$\mathtt{f}_s^{\mathtt{Fr}}(\mathtt{a_from}) = \mathtt{a},\ \mathtt{f}_t^{\mathtt{Fr}}(\mathtt{a_from}) = \mathtt{n}, \quad (29)$$

$$\mathtt{f}_s^{\mathtt{To}}(\mathtt{a_to}) = \mathtt{a},\ \mathtt{f}_t^{\mathtt{To}}(\mathtt{a_to}) = \mathtt{n}, \quad (30)$$

$$\mathtt{f}_{type}(\mathtt{tok}) = \mathtt{p},\ \mathtt{f}_{val}(\mathtt{tok}) = \mathbb{N}_0, \quad (31)$$

$$\mathtt{f}_{DT}(i) = \mathbb{N}_0\ \forall i \in \mathcal{U}_{\mathrm{D}} \quad (32)$$

This formal metamodel $\mathtt{M}_{PN}$ conforms to all constraints (13)–(19) and describes the formal language $\mathcal{PN}$ introduced in Example 1. Their subordination prompts a generic procedure on how to deduce the latter from the former. In Table 1,

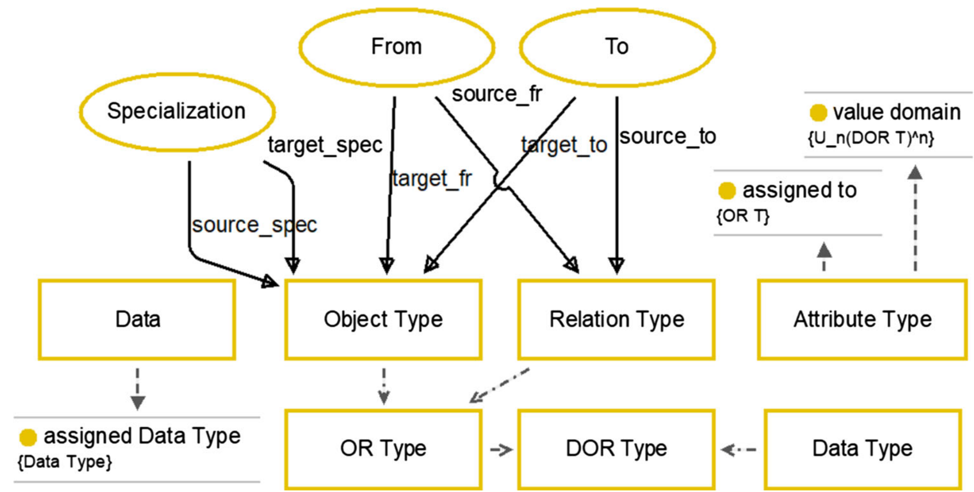

Fig. 6 The metamodel of M2FOL (notation see Fig. 2)

we present this procedure as an algorithm. In the right column, the algorithm is exemplified on the metamodel of Petri Nets. Compare the result to Example 1.

4.3 Meta-perspective on M2FOL

Finally, we formalize the metamodel of M2FOL as M2FOL model. The graphical metamodel is depicted in Fig. 6.

Example 4 Metamodel of M2FOL
The M2FOL metamodel contains seven objects of type OT={o(*bject*)t(*ype*), r(*elation*)t(*ype*), a(*ttribute*)t(*ype*), d(*ata*)t(*ype*), d(*ata*), ort(*ype*), dort(*ype*)}, three objects of type RT={spec, fr(*om*), to}, three objects of type AT= {val(*ue*)_dom(*ain*), ass(*igned*)_to, ass(*igned*)_d(*ata*)t(*ype*), many objects of type DT $=$ {dort$^i\forall i$, $\bigcup_i$(dort)^i} (not visible in the graphical metamodel), four relations of type Spec={ot $<$ ort, rt $<$ ort, ort $<$ dort, dt $<$ dort}, three relations of type From={source_spec, source_to, source_fr}, as well as three relations in To={target_spec, target_to, target_fr}, furthermore 26 assignments of source and target objects, attribute owning types and attribute value types.

$$f_s^{To}(\texttt{target_spec}) = \texttt{spec}, \tag{33}$$
$$f_t^{To}(\texttt{target_spec}) = \texttt{ot}, \tag{34}$$
$$f_s^{Fr}(\texttt{source_spec}) = \texttt{spec}, \tag{35}$$
$$f_t^{Fr}(\texttt{source_spec}) = \texttt{ot}, \tag{36}$$
$$f_s^{To}(\texttt{target_fr}) = \texttt{fr}, \tag{37}$$
$$f_t^{To}(\texttt{target_fr}) = \texttt{ot}, \tag{38}$$
$$f_s^{Fr}(\texttt{source_fr}) = \texttt{fr}, \tag{39}$$
$$f_t^{Fr}(\texttt{source_fr}) = \texttt{rt}, \tag{40}$$
$$f_s^{To}(\texttt{target_to}) = \texttt{to}, \tag{41}$$
$$f_t^{To}(\texttt{target_to}) = \texttt{ot}, \tag{42}$$
$$f_s^{Fr}(\texttt{source_to}) = \texttt{to}, \tag{43}$$
$$f_t^{Fr}(\texttt{source_to}) = \texttt{rt}, \tag{44}$$
$$f_s^{Spec}(\texttt{ot} < \texttt{ort}) = \texttt{ot}, \tag{45}$$
$$f_t^{Spec}(\texttt{ot} < \texttt{ort}) = \texttt{ort}, \tag{46}$$
$$f_s^{Spec}(\texttt{rt} < \texttt{ort}) = \texttt{rt}, \tag{47}$$
$$f_t^{Spec}(\texttt{rt} < \texttt{ort}) = \texttt{ort}, \tag{48}$$
$$f_s^{Spec}(\texttt{ort} < \texttt{dort}) = \texttt{ort}, \tag{49}$$
$$f_t^{Spec}(\texttt{ort} < \texttt{dort}) = \texttt{dort}, \tag{50}$$
$$f_s^{Spec}(\texttt{dt} < \texttt{dort}) = \texttt{dt}, \tag{51}$$
$$f_t^{Spec}(\texttt{dt} < \texttt{dort}) = \texttt{dort}, \tag{52}$$
$$f_{type}(\texttt{ass_dt}) = \texttt{d},\ f_{val}(\texttt{ass_dt}) = \texttt{dt} \tag{53}$$
$$f_{type}(\texttt{ass_to}) = \texttt{at},\ f_{val}(\texttt{ass_to}) = \texttt{ort}, \tag{54}$$
$$f_{type}(\texttt{val_dom}) = \texttt{at}, \tag{55}$$
$$f_{val}(\texttt{val_dom}) = \bigcup_i (\texttt{dort})^i \tag{56}$$

On the one hand, the construct above is itself a model expressed in the language M2FOL. On the other hand, this metamodel defines M2FOL as a meta^{2}model. With the algorithm presented above, we deduce Definition 3 from Example 4. So we conclude that the proposed modeling language M2FOL for metamodels is self-describing and thereby complete the formalization of the four-layer metamodeling stack.

4.4 Pinpointing the approach in the language definition hierarchy

In Fig. 7, the language definition hierarchy and model hierarchy adapted from Mayr and Thalheim [47] are depicted. We use this hierarchy to pinpoint the definitions and examples presented so far and illustrate the big picture of the approach with all its interdependencies. Definitions and examples of

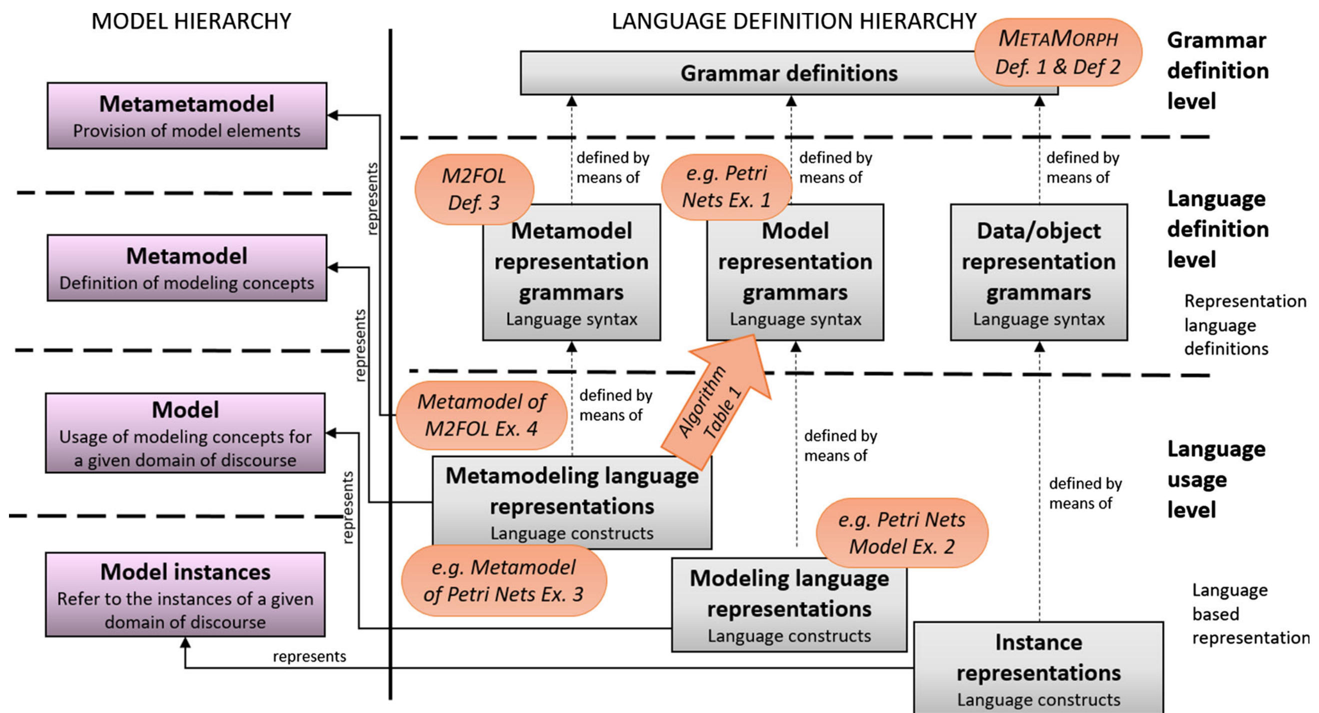

Fig. 7 Definitions and examples of the presented approach are pinpointed (orange rounded rectangles) in the big picture of the language definition hierarchy and model hierarchy outlined in [47]

the paper at hand are inserted as orange rounded rectangles in the figure to disclose their role in the language definition hierarchy. Note that in Fig. 7 there is an explicit distinction between the actual model and the language construct representing the model, e.g., the modeling language representation of a model or the metamodeling language representation of a metamodel. Although models, metamodels, metametamodels, etc., belong to different levels in the model hierarchy, their language representations reside on the same level in the language definition hierarchy, namely on the *language usage level* as they are model representations defined by means of a modeling language. On this level, we find the Petri Nets model representation from Example 2 and the representation of the Petri Nets metamodel from Example 3. The metamodel of `M2FOL` from Example 4 is an example of a meta^{2}modeling language representation.

All these model representations are language constructs defined by means of a language that is defined on the next higher level, the *language definition level*. On this level, we find representation grammars specifying the language syntax. The model representation grammar for the Petri Nets model mentioned above is the Petri Nets language definition from Example 1. The metamodel representation grammar for the M2FOL model mentioned above is the M2FOL language definition from Definition 3.

Again the means to define these representation grammars resides on the next higher level in the language definition hierarchy, namely on the *grammar definition level*. The authors of [47] name versions of EBNF as an example for elements on this level. In the presented approach, we use the METAMORPH formalism for grammar definition comprising Definition 1 and Definition 2 precisely defining how to specify a modeling language or, in other words, a representation grammar.

As a metamodel defines a modeling language (see Fig. 5), we present in Table 1 an algorithm to deduce the model representation grammar (located at the *language definition level*) from a metamodeling language representation (located at the *language usage level*). Therefore, the algorithm builds a bridge between levels. It is depicted as an arrow in Fig. 7.

On the *language usage level*, we also find the metamodel of the metamodeling language `M2FOL` (Example 4), therefore being a representation of a metametamodel. As this representation was itself defined by means of `M2FOL`, this shows that `M2FOL` is a metamodel representation grammar as well as a metametamodel representation grammar and therefore suffices to complete the formalization of the four-layer metamodeling stack.

5 Potential and benefits of formalized conceptual modeling languages

In this section, we exemplify the integrability of METAMORPH and give an outlook to several research topics

potentially benefiting from formalizing conceptual modeling languages with the proposed formalism—these are modeling with power types, language interleaving and consistency, operations on models, and translators of platform-independent formalizations to platform-specific code. For this purpose, we make use of established concepts from formal language theory. For reasons of brevity, we will not discuss all topics in detail and will restrict to an extensive elaboration only for the first two topics of interest.

5.1 Modeling with power types

Power types are an advanced tool in conceptual modeling and address issues arising when modeling, e.g., biological taxonomic relations like birds and bird species [53] or dogs and breeds [31], issues that motivated the field of multi-level modeling [26]. Originally stemming from the object-oriented programming domain [52], power types are a means to describe an instantiation dependency (also named classification) between elements in a model. It is one of the most common techniques in multi-level modeling [15].

To be precise, a type is the power type of another type, the base type, if all its instances are specializations of the base type [15]. Power types are based on the mathematical concept of a powerset [31]. Depending on the concrete requirements on the elements of the power type—forming the full powerset, a partition of the base set, disjoint subsets, etc.—the realization of the power type concept shows slightly different behavior [15]. In this outlook, we adopt the most general variant of the concept and allow arbitrary subsets of the powerset to constitute the power type.

In the following, we will show that the concept of power types can be canonically integrated into the proposed formalism. With this extension METAMORPH allows for tackling situations core to the multi-level modeling approach. This proves the ease of extending METAMORPH with relevant concepts from conceptual modeling and therefore underpins the suitability of formal languages as underlying structural theory. This extension shows again the capability of formalized models to capture information that has no canonical counterpart in the graphical representation of the model.

5.1.1 Extending METAMORPH with the power type concept

The power type pattern defines a relation on the object types of a language similar to the specialization relation. A type P is a power type of another type B, the base type, if all instances of P are sets of instances of B, i.e., in a model the universe of elements $\mathcal{U}_P$ is a subset of the powerset $\wp(\mathcal{U}_B)$. This dependency of object types induces a new binary relation $\in_O \subseteq \mathcal{S}_O \times \mathcal{S}_O$. Unlike the specialization relation $<_O$ it is not an (strict partial) order relation, as it does not fulfill transitivity, but it has to be anti-reflexive and circle-free.

The power type pattern allows for the assignment of new attributes to categorized objects of its base type. This means that an element b of the base type receives new features by the membership in a concrete element p of the power type, which is a set. Therefore, we define these attributes as triples (b, p, val) between the element, its power type element, and the actual value of the attribute. To realize this, we have to prepare some data types to enable the definition of an attribute and its value domain on model-level because they are not known at the language definition level. First, we need a set $\mathbf{T}_{all}$ in $\mathcal{S}_D$ containing all possible attribute values offered by the language. The value domain of an added attribute is then a set of elements in $\mathbf{T}_{all}$, i.e., the value domain is an element in the powerset $\wp(\mathbf{T}_{all})$. We define the assignment of this domain to a power type element p of type $\mathbf{O}_P$ via a function $F_{\mathbf{O}_P} : \mathbf{O}_P \rightarrow \wp(\mathbf{T}_{all})$. Considering the attribute value assignment to a base element b via a relation $R \ni (b, p, val)$, we do not know at the language definition level the concrete domain val belongs to. Therefore, we define val as an element in $\mathbf{T}_{all}$. The requirement that val is contained in the assigned attribute domain of p has to be ensured by a constraint: $val \in F_{\mathbf{O}_P}(p)$.

In the following, we present the extended Definition 1, where the new parts are italicized. Note that we only admit a single new attribute per power type. The definition of arbitrary many attributes is possible by using the product type $\bigcup_i \wp(\mathbf{T}_{all})^i$ as codomain of $F_{\mathbf{O}_P}$ and $\bigcup_i (\mathbf{T}_{all})^i$ as third component in R. We skip a detailed explanation of this more generic case to limit the complexity of the definition for this outline of potentials of the approach. We also allow only values already existing in the language definition for the attribute domain.

Definition 4 (*extending Definition 1*) A (formal) modeling language $\mathcal{L}$ including the concept of power types consists of a typed signature Σ with $\Sigma = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}$ and a set $\mathbf{C}$ of sentences in $\mathcal{L}$ for the constraints, where:

- $\mathcal{S}$ is a set of types, which can be further divided into three disjoint subsets $\mathcal{S}_O$, $\mathcal{S}_R$, and $\mathcal{S}_D$ for object types, relation types and data types;
  - the type set $\mathcal{S}_O$ is strictly partially ordered with order relation $<_O \subseteq \mathcal{S}_O \times \mathcal{S}_O$ to indicate the specialization relation between the corresponding object types;
  - *the type set $\mathcal{S}_O$ is furthermore structured by the anti-reflexive and circle-free power type relation $\in_O \subseteq \mathcal{S}_O \times \mathcal{S}_O$ to indicate instantiation between the base type and its power type;*
  - the type set $\mathcal{S}_D$ can contain simple types $\mathbf{T}$ for value domains of single-value attributes, or product types $\mathbf{T}' = \mathbf{T}_1 \times \mathbf{T}_2 \times \cdots \times \mathbf{T}_n$ and unions thereof for value

domains of n-ary multi-value attributes ($n > 1$), where the ith value is *of type* $\mathbf{T}_i \in \mathcal{S}_D \cup \mathcal{S}_O \cup \mathcal{S}_R \setminus \{\mathbf{T}_{all}, \wp(\mathbf{T}_{all})\}$;
  - *the type set* $\mathcal{S}_D$ *additionally contains a type* $\mathbf{T}_{all}$ *and its powerset* $\wp(\mathbf{T}_{all})$ *where* $\mathbf{T}_{all}$ *is the union of all object types, relation types, simple and product types* $\mathbf{T}_{all} = \bigcup_{\mathbf{T} \in \mathcal{S}'} \mathbf{T}$, *with* $\mathcal{S}' = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D \setminus \{\mathbf{T}_{all}, \wp(\mathbf{T}_{all})\}$;

- $\mathcal{F}$ is a set of typed function symbols such that:
  - for each relation type $\mathbf{R}$ in $\mathcal{S}_R$ there exist two function symbols $F_s^{\mathbf{R}}$ and $F_t^{\mathbf{R}}$ with domain type $\mathbf{R} \in \mathcal{S}_R$ and codomain type $\mathbf{O}_s, \mathbf{O}_t \in \mathcal{S}_O$ assigning the source and target object types to a relation;
  - for each single-value attribute $\mathbf{A}$ of an object or relation type $\mathbf{T}$ there exists a function symbol $F^{\mathbf{A}}$ with domain type $\mathbf{T}$ and codomain type a simple type in $\mathcal{S}_D$ or an element in $\mathcal{S}_O$ or $\mathcal{S}_R$ assigning the simple data type or referenced object type or relation type to the attribute;
  - for each multi-value attribute $\mathbf{A}$ of an object or relation type $\mathbf{T}$ there exists a function symbol $F^{\mathbf{A}}$ with domain type $\mathbf{T}$ and codomain type a product type in $\mathcal{S}_D$ and unions thereof;
  - *for each power type relation* $\mathbf{O}_B \in_O \mathbf{O}_P$ *the set* $\mathcal{F}$ *contains a function symbol* $F_{\mathbf{O}_P} : \mathbf{O}_P \rightarrow \wp(\mathbf{T}_{all})$ *assigning a set of possible values to the additional attribute added to the power type on model-level;*

- $\mathcal{R}$ is a set of typed relation symbols such that:
  - *for each power type relation* $\mathbf{O}_B \in_O \mathbf{O}_P$ *the set* $\mathcal{R}$ *contains a relation symbol* $\in_{BP} \subseteq \mathbf{O}_B \times \mathbf{O}_P$ *to enable the check of membership of base element and power type element on model-level;*
  - *for each power type relation* $\mathbf{O}_B \in_O \mathbf{O}_P$ *the set* $\mathcal{R}$ *contains a relation symbol* $R_{BP} \subseteq \mathbf{O}_B \times \mathbf{O}_P \times \mathbf{T}_{all}$ *assigning a value to the added attribute assigned to an element in* $\mathbf{O}_B$ *by its membership in* $\mathbf{O}_P$;
  - $\mathcal{R}$ *contains a relation symbol* $\in_{val} \subseteq \mathbf{T}_{all} \times \wp(\mathbf{T}_{all})$ *with the usual containment semantics of elements and sets;*

- $\mathcal{C}$ is a set of typed constants to specify the possible values c_i of a simple type $\mathbf{T} \in \mathcal{S}_D$ of the attributes;
- the set $\mathbf{C}$ is a set of sentences in $\mathcal{L}$ constraining the possible models, also called the postulates of the language. *To ensure the proper behavior of the power type relation we need several constraints for each pair of related types* $\mathbf{O}_B$ *base type and* $\mathbf{O}_P$ *power type:*
  - *A language comprising a power type relation* $\mathbf{O}_B \in_O \mathbf{O}_P$ *must ensure that a value is assigned to the pair* (b, p) *of a base type element* b *and power type element* p *iff* b *and* p *are in a power type relation:*

$$\forall b \in \mathbf{O}_B, p \in \mathbf{O}_P \quad (b \in_{BP} p \Leftrightarrow \exists x \in \mathbf{T}_{all} \; ((b, p, x) \in R_{BP}) \tag{57}$$

  - *To conform to the value domain of an additional attribute induced by the power type relation* $\mathbf{O}_B \in_O \mathbf{O}_P$ *we furthermore need a constraint binding the type of the assigned value* $val \in \mathbf{T}_{all}$ *of the relation symbol* $R_{BP} \subset \mathbf{O}_B \times \mathbf{O}_P \times \mathbf{T}_{all}$ to $F_{\mathbf{O}_P}(p)$:

$$\forall x \in \mathbf{T}_{all}, b \in \mathbf{O}_B, p \in \mathbf{O}_P \quad (((b, p, x) \in R_{BP}) \implies x \in_{val} F_{\mathbf{O}_P}(p)) \tag{58}$$

  - *The assigned value* $x \in \mathbf{T}_{all}$ *to the tuple of base type element and power type element must be unique:*

$$\forall x, y \in \mathbf{T}_{all} \; ((\exists b \in \mathbf{O}_B, p \in \mathbf{O}_P \; (((b, p, x) \in R_{BP}) \wedge ((b, p, y) \in R_{BP}))) \implies x = y) \tag{59}$$

In the following, we present the extended Definition 2 of a model, where the new parts are italicized:

Definition 5 (*extending Definition* 2) A model $\mathcal{M}$ of a language $\mathcal{L}$ with typed signature $\Sigma = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}$ including a concept of power types is an $\mathcal{L}$-structure conforming to the language constraints $\mathbf{C}$, i.e., $\mathcal{M}$ consists of

- a universe $\mathcal{U}$ of typed elements respecting the type hierarchy and power type relation, that is
  - for each $\mathbf{T}$ in $\mathcal{S}$ there exists a set $\mathcal{U}_{\mathbf{T}} \subseteq \mathcal{U}$ and $\mathcal{U} = \bigcup_{\mathbf{T} \in \mathcal{S}} \mathcal{U}_{\mathbf{T}}$;
  - all sets $\mathcal{U}_{\mathbf{T}}$ for $T \in \mathcal{S}_O \cup \mathcal{S}_R$ have to be pairwise disjoint except for sets $\mathcal{U}_{\mathbf{O}_1}$ and $\mathcal{U}_{\mathbf{O}_2}$ with $\mathbf{O}_1, \mathbf{O}_2 \in \mathcal{S}_O$ where $\mathbf{O}_1 <_O \mathbf{O}_2$. In this case $\mathcal{U}_{\mathbf{O}_1}$ must be a subset of $\mathcal{U}_{\mathbf{O}_2}$, i.e., $\mathcal{U}_{\mathbf{O}_1} \subseteq \mathcal{U}_{\mathbf{O}_2}$;
  - all sets $\mathcal{U}_{\mathbf{T}}$ with $\mathbf{T} = \mathbf{T}_1 \times \mathbf{T}_2 \times \cdots \times \mathbf{T}_n$ a product type in $\mathcal{S}_D$ consist of tuples $(x_1, x_2, \ldots x_n) \in \mathcal{U}_{\mathbf{T}_1} \times \mathcal{U}_{\mathbf{T}_2} \times \cdots \times \mathcal{U}_{\mathbf{T}_n}$;
  - *all elements* $\mathcal{U}_{\mathbf{O}_P}$ *with* $\mathbf{O}_B, \mathbf{O}_P \in \mathcal{S}_O$ *and* $\mathbf{O}_B \in_O \mathbf{O}_P$ *are sets of elements of* $\mathcal{U}_{\mathbf{O}_B}$. *This means that* $\mathcal{U}_{\mathbf{O}_P}$ *is a set of sets* $\subseteq \mathcal{U}_{\mathbf{O}_B}$ *and therefore a subset of the powerset* $\wp(\mathcal{U}_{\mathbf{O}_B})$.
- an interpretation of the function symbols in $\mathcal{L}$, i.e., for each function symbol $F \in \mathcal{F}$ with domain type $\mathbf{T}_1 \times \ldots \times \mathbf{T}_n$ and codomain type $\mathbf{T}$ a function $f : \mathcal{U}_{\mathbf{T}_1} \times \ldots \times \mathcal{U}_{\mathbf{T}_n} \rightarrow \mathcal{U}_{\mathbf{T}}$;
- an interpretation of the relation symbols in $\mathcal{L}$, i.e., for each relation symbol $R \in \mathcal{R}$ with domain type $\mathbf{T}_1 \times \ldots \times \mathbf{T}_m$ a relation $r \subseteq \mathcal{U}_{\mathbf{T}_1} \times \ldots \times \mathcal{U}_{\mathbf{T}_m}$;

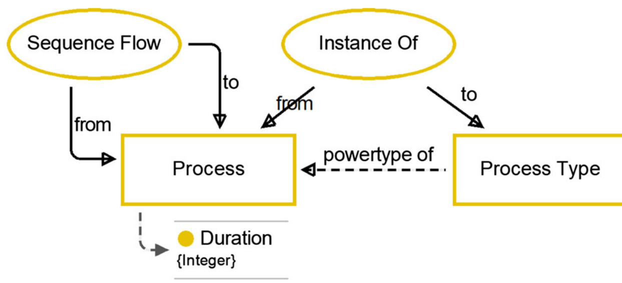

Fig. 8 Simple metamodel of a customizable process modeling language (notation see Fig. 2)

- for each simple type $\mathbf{T} \in \mathcal{S}_D$ and constant $C \in \mathcal{C}$ of type $\mathbf{T}$ an interpretation $c \in \mathcal{U}_\mathbf{T}$;
- for each constraint ϕ in $\mathbf{C}$ the model $\mathcal{M}$ satisfies ϕ, i.e., $\mathcal{M} \models \phi$, in particular $\mathcal{M}$ satisfies the constraints for the proper behavior ot the power types.

The power type relation is not per se visually represented just as a usual relation type in $\mathcal{S}_R$ is, but we can include a visual counterpart by inducing a supplementary relation type in $\mathcal{S}_R$ corresponding to $\in_{BP}$ via constraints, see Example 5.

5.1.2 Case study

The concept of power types in METAMORPH will be demonstrated on a case study of a very simple customizable process modeling language.

Example 5 Customizable process modeling language
In Fig. 8, we see the metamodel of a process modeling language that allows for the creation of enterprise specific process types to classify processes. The metamodel comprises two object types **Process** (**P**), and **ProcessType** (**PT**) constituting $\mathcal{S}_O$. Thereby, **ProcessType** is a power type of **Process**: **Process** $\in_O$ **ProcessType**. The language does not comprise specialization relations on its object types. Between process elements we define a relation **SequenceFlow** (**SF**) to represent the sequence of process steps. Furthermore, we include a relation type **InstanceOf** (**IO**) element of $\mathcal{S}_R$ visualizing the power type relation. We include types $\mathbb{N}_0$ and **Boolean** and specify constants in $\mathcal{C} = \{0, 1, 2, \ldots\} \cup \{true, false\}$ of type $\mathbb{N}_0$ and **Boolean**, respectively. Furthermore we include $\mathbf{T}_{all} = \bigcup \mathbf{T}$ running over all $\mathbf{T}$ in $\mathcal{S}_D \cup \mathcal{S}_O \cup \mathcal{S}_R \setminus \{\mathbf{T}_{all}, \wp(\mathbf{T}_{all})\}$ and its powerset $\wp(\mathbf{T}_{all})$. The set $\mathcal{S}$ of types is then the union $\mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D$={**Process**, **ProcessType**, **SequenceFlow**, **InstanceOf**, $\mathbb{N}_0$, **Boolean**, $\mathbf{T}_{all}$, $\wp(\mathbf{T}_{all})$}. For the relation types **SequenceFlow** and **InstanceOf**, we have to specify the source and target object types by introducing four function symbols $F_s^{\mathbf{SF}}$ and $F_t^{\mathbf{SF}}$ both with domain **SequenceFlow** and codomain **Process**, as well as $F_s^{\mathbf{IO}}$ and $F_t^{\mathbf{IO}}$ both with domain **InstanceOf**. The codomain of $F_s^{\mathbf{IO}}$ is **Process** and the codomain of $F_t^{\mathbf{IO}}$ is **ProcessType**. A process owns an attribute **Duration** with function symbol $F^{\mathbf{Dur}}$: **Process** $\rightarrow \mathbb{N}_0$, assigning the execution time in minutes to the process. The power type relation between **Process** and **ProcessType** further requires a function symbol $F_{\mathbf{PT}}$: **ProcessType** $\rightarrow \wp(\mathbf{T}_{all})$ to allow the definition of a new attribute on model-level and a relation symbol $R_{\mathbf{PT}} \subseteq$ **Process** $\times$ **ProcessType** $\times\, \mathbf{T}_{all}$ for the attribute value assignment. Also, two relation symbols to check containment $\in_{val}$ and $\in_{\mathbf{PPT}}$ are needed.

The signature of the language looks as follows:

$$\Sigma = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\},\ \mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D \tag{60}$$

$$\mathcal{S}_O = \{\mathbf{Process},\ \mathbf{ProcessType}\}, \tag{61}$$

$$\mathbf{Process} \in_O \mathbf{ProcessType} \tag{62}$$

$$\mathcal{S}_R = \{\mathbf{InstanceOf},\ \mathbf{SequenceFlow}\}, \tag{63}$$

$$\mathcal{S}_D = \{\mathbb{N}_0, \mathbf{Boolean}, \mathbf{T}_{all}, \wp(\mathbf{T}_{all})\}, \tag{64}$$

$$\begin{aligned}\mathcal{F} = \{&F_s^{\mathbf{IO}} : \mathbf{InstanceOf} \rightarrow \mathbf{Process},\\ &F_t^{\mathbf{IO}} : \mathbf{InstanceOf} \rightarrow \mathbf{ProcessType},\\ &F_s^{\mathbf{SF}} : \mathbf{SequenceFlow} \rightarrow \mathbf{Process},\\ &F_t^{\mathbf{SF}} : \mathbf{SequenceFlow} \rightarrow \mathbf{Process},\\ &F_{\mathbf{PT}} : \mathbf{ProcessType} \rightarrow \wp(\mathbf{T}_{all}),\\ &F^{\mathbf{Dur}} : \mathbf{Process} \rightarrow \mathbb{N}_0\}\end{aligned} \tag{65}$$

$$\begin{aligned}\mathcal{R} = \{&\in_{\mathbf{PPT}} \subseteq \mathbf{Process} \times \mathbf{ProcessType},\\ &\in_{val} \subset \mathbf{T}_{all} \times \wp(\mathbf{T}_{all}),\\ &R_{\mathbf{PT}} \subseteq \mathbf{Process} \times \mathbf{ProcessType} \times \mathbf{T}_{all}\},\end{aligned} \tag{66}$$

$$\begin{aligned}&\mathcal{C} = \{0, 1, 2, \ldots\}, \cup \{true, false\}\\ &\quad \text{of type } \mathbb{N}_0 \text{ and } \mathbf{Boolean}.\end{aligned} \tag{67}$$

We need several constraints to ensure the intended behavior of the power type relation on **ProcessType** and **Process**. For this simple example, we do not define any further constraints.

First of all, we bind the existence of the graphical **InstanceOf** relation to the power type relation between **ProcessType** and **Process**:

$$\begin{aligned}&\forall p \in \mathbf{P},\ pt \in \mathbf{PT}\ (p \in_{\mathbf{PPT}} pt \Leftrightarrow\\ &\quad \exists r \in \mathbf{IO}\ F_s^{\mathbf{IO}}(r) = p \wedge F_t^{\mathbf{IO}}(r) = pt)\end{aligned} \tag{68}$$

For each process p that is element in the power type pt of type **ProcessType** there must be a value x assigned to the attribute added to p by pt:

$$\begin{aligned}&\forall p \in \mathbf{P},\ pt \in \mathbf{PT}\\ &\quad (p \in_{\mathbf{PPT}} pt \Leftrightarrow \exists x \in \mathbf{T}_{all}\ ((p, pt, x) \in R_{\mathbf{PT}})\end{aligned} \tag{69}$$

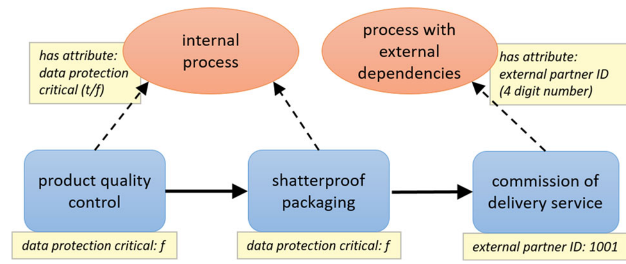

Fig. 9 Example model of the customizable process language

The assigned value must be an element of the domain specified by $F_{\mathbf{PT}}(pt)$:

$$\forall x \in \mathbf{T}_{all}, p \in \mathbf{P}, pt \in \mathbf{PT} \\ (((p, pt, x) \in R_{\mathbf{PT}}) \implies x \in F_{\mathbf{PT}}(pt)) \tag{70}$$

The assigned value $x \in \mathbf{T}_{all}$ to the tuple (p, pt) must be unique:

$$\forall x, y \in \mathbf{T}_{all} \ (\exists p \in \mathbf{P}, pt \in \mathbf{PT} \\ (((p, pt, x) \in R_{\mathbf{PT}}) \wedge ((p, pt, y) \in R_{\mathbf{PT}})) \\ \implies x = y) \tag{71}$$

We demonstrate the use of power types on model-level on a simple process of a glass manufacturer representing the last steps of the production process: from quality control to shipping. For the notation we borrow the symbols of BPMN for processes and sequence flows and extend it with a dashed arrow for the **InstanceOf** relation and ovals for **ProcessType**. The newly defined attributes are added via notes, as this information usually has no place in a graphical model.

Example 6 A customized process of a glass manufacturer In Fig. 9, we see the last few steps of the production process of a glass manufacturer. The enterprise specific process types are *internal process* **(ipr)** and *process with external dependencies* **(ped)**. The three depicted processes concern the *product quality control* **(qc)**, the *shatterproof packaging* **(sp)**, and the *commission of delivery service* **(ds)**. The process types are sets of processes, i.e., elements of the powerset $\wp(\mathbf{Process}) = \{\{\}, \{qc\}, \{sp\}, \{ds\}, \{qc, sp\}, \{qc, ds\}, \{sp, ds\}, \{qc, sp, ds\}\}$. In this concrete case they are $\mathbf{ipr} = \{\mathbf{qc}, \mathbf{sp}\}$, and $\mathbf{ped} = \{\mathbf{ds}\}$. This containment, that is not visible in the model, is reflected in the (graphically represented) relation type **InstanceOf** induced by constraint (68) of the language. In this case this means that each listed process instance is source of one relation $\mathbf{io_{qc}}$, $\mathbf{io_{sp}}$, and $\mathbf{io_{ds}}$ of type **InstanceOf**. Target of the first two elements $\mathbf{io_{qc}}$, and $\mathbf{io_{sp}}$ is **ipr**, target of the third one $\mathbf{io_{ds}}$ is **ped**. For each power type a new attribute is defined. **Internal processes** are classified according to whether they are data protection critical, captured in a Boolean value. **Processes with external dependencies** include an ID of the external partner involved. The ID is a 4 digit number. The function $F_{\mathbf{PT}}$ specifies these value domains of the newly defined attributes: $f_{\mathbf{PT}}(\mathbf{ipr}) = \{true, false\}$ and $f_{\mathbf{PT}}(\mathbf{ped}) = \{1000, 1001, \ldots, 9999\}$. The concrete values assigned to the instances of the power type **ProcessType** are then defined via the relation symbol $R_{\mathbf{PT}}$: **product quality control** is not data protection critical, $(\mathbf{qc}, \mathbf{ipr}, false) \in R_{\mathbf{PT}}$, and neither is **shatterproof packaging**, $(\mathbf{sp}, \mathbf{ipr}, false) \in R_{\mathbf{PT}}$. The partner ID of the delivery service is 1001, $(\mathbf{ds}, \mathbf{ped}, 1001) \in R_{\mathbf{PT}}$. These assignments fulfill the postulates for existence (69) and uniqueness of the assigned value (71), as well as the conformance to the assigned attribute domain $f_{\mathbf{PT}}$ (70).

The processes are connected by two **SequenceFlow** relations **flow1**, and **flow2** in the same order as appearing in the list above. Furthermore, the duration of the processes is not visible in the graphical model but specified in the formal model via the function $F^{\mathbf{Dur}}$: $f^{\mathbf{Dur}}(\mathbf{qc}) = 15$, $f^{\mathbf{Dur}}(\mathbf{sp}) = 10$, and $f^{\mathbf{Dur}}(\mathbf{ds}) = 5$.

The complete, formalized model looks as follows:

$$\mathcal{U}_{\mathbf{P}} = \{\textbf{product quality control (qc)}, \tag{72}$$
$$\textbf{shatterproof packaging (sp)}, \tag{73}$$
$$\textbf{commission of delivery service (ds)}\} \tag{74}$$
$$\mathcal{U}_{\mathbf{PT}} = \{\textbf{internal process (ipr)}, \tag{75}$$
$$\textbf{process with external dependencies (ped)}\} \tag{76}$$
$$\textbf{ipr=\{qc, sp\}, ped=\{ds\}} \tag{77}$$
$$\mathcal{U}_{\mathbb{N}_0} = \{0, 1, 2, \ldots\} \tag{78}$$
$$\mathcal{U}_{\mathbf{Boolean}} = \{true, false\} \tag{79}$$
$$\mathcal{U}_{\mathbf{IO}} = \{\mathbf{io_{qc}}, \mathbf{io_{sp}}, \mathbf{io_{qc}}\} \tag{80}$$
$$f_s^{\mathbf{IO}}(\mathbf{io_{qc}}) = \mathbf{qc},\ f_t^{\mathbf{IO}}(\mathbf{io_{qc}}) = \mathbf{ipr}, \tag{81}$$
$$f_s^{\mathbf{IO}}(\mathbf{io_{sp}}) = \mathbf{sp},\ f_t^{\mathbf{IO}}(\mathbf{io_{sp}}) = \mathbf{ipr}, \tag{82}$$
$$f_s^{\mathbf{IO}}(\mathbf{io_{ds}}) = \mathbf{ds},\ f_t^{\mathbf{IO}}(\mathbf{io_{ds}}) = \mathbf{ped}. \tag{83}$$
$$\mathcal{U}_{\mathbf{SF}} = \{\mathbf{flow1}, \mathbf{flow2}\} \tag{84}$$
$$f_s^{\mathbf{SF}}(\mathbf{flow1}) = \mathbf{qc},\ f_t^{\mathbf{SF}}(\mathbf{flow1}) = \mathbf{sp}, \tag{85}$$
$$f_s^{\mathbf{SF}}(\mathbf{flow2}) = \mathbf{sp},\ f_t^{\mathbf{SF}}(\mathbf{flow2}) = \mathbf{ds}, \tag{86}$$
$$f^{\mathbf{Dur}}(\mathbf{qc}) = 15,\ f^{\mathbf{Dur}}(\mathbf{sp}) = 10,\ f^{\mathbf{Dur}}(\mathbf{ds}) = 5, \tag{87}$$
$$f_{\mathbf{PT}}(\mathbf{ipr}) = \{true, false\}, \tag{88}$$
$$f_{\mathbf{PT}}(\mathbf{ped}) = \{1000, 1001, \ldots, 9999\}. \tag{89}$$
$$R_{\mathbf{PT}} = \{(\mathbf{qc}, \mathbf{ipr}, false), \\ (\mathbf{sp}, \mathbf{ipr}, false), (\mathbf{ds}, \mathbf{ped}, 1001)\}, \tag{90}$$
$$\mathcal{U}_{\mathbf{T}_{all}} = \{\mathbf{qc}, \mathbf{sp}, \mathbf{ds}, \mathbf{ipr}, \mathbf{ped}, \mathbf{io_{qc}}, \mathbf{io_{sp}}, \mathbf{io_{qc}}, \\ \mathbf{flow1}, \mathbf{flow2}, true, false, 0, 1, 2, \ldots\} \tag{91}$$

The universe of $\mathcal{U}_{\wp(\mathbf{T}_{all})}$ consists of all subsets of $\mathcal{U}_{\mathbf{T}_{all}}$.

5.2 Language interleaving and consistency

Models are means to manage information in highly complex systems in business modeling, in software engineering, and many other fields. The solution to cope with complexity is often seen in the distribution and fragmentation of information between different models or views possibly in different modeling languages, thereby raising the issue of keeping the models consistent [2,39].

In the following, we demonstrate that language interleaving and the definition of consistency constraints can be easily realized in formalized conceptual modeling languages. Depending on the initial situation we can distinguish top-down approaches, where a newly defined or existing language is segmented in several sublanguages or views, and bottom-up approaches, where existing languages are interleaved and their metamodels are amalgamated and equipped with additional constraints [49]. We will discuss the first approach briefly and exemplify the second one in a case study.

5.2.1 Top-down approach

Expressed in our formalism the top-down approach means to restrict the signature Σ of a given language $\mathcal{L}$ to subsets Σ_1 and Σ_2 of the signature. When working in one view, i.e., with a sublanguage $\mathcal{L}|_{\Sigma_i}$, we are restricted only to the types appearing in Σ_i. Note that we also have to remove relation types if their source or target object type was excluded from the signature as well as for attribute types if their source type or value type was excluded. All constraints considering unavailable types have to be removed. Note also that the signatures Σ_i of the sublanguages do not have to be disjoint. While restricting to a sublanguage $\mathcal{L}|_{\Sigma_i}$ and thereby restricting to a concrete view on a system under study we are still interesting in the model as a whole. So we assume that for each view of $\mathcal{L}|_{\Sigma_i}$ there exist correlated models in the other views $\mathcal{L}|_{\Sigma_j}$ being dependent on each other. Therefore, we need pairwise constraints between the possible views always considering the signatures of the two relevant languages. If the signatures are not disjoint, these constraints contain bijections of elements with a common type to keep the shared structure consistent.

5.2.2 Bottom-up approach

Expressed in our formalism the bottom-up approach means to fusion the signatures of two given languages $\mathcal{L}_1$ and $\mathcal{L}_2$. The interleaving of models and their language reaches from simply referencing elements in other models to a highly dependent content and structure of models in both directions. There exist different techniques to link conceptual modeling languages [2]. There are also different techniques in the field of logic on how to combine formal languages, e.g., [4,44]. The presented attempt is based on the approach proposed in [4].

Uniting two given languages $\mathcal{L}_1$ and $\mathcal{L}_2$ requires uniting their signatures Σ_1 and Σ_2. When doing so, we have to take care of types $\mathbf{T}$ and other symbols occurring in both languages. To stay compatible with existing models we keep coincident object types $\mathbf{T}$ and rename them to $\mathbf{T}_1$ and $\mathbf{T}_2$. Furthermore, we introduce new function symbols $i : \mathbf{T}_1 \to \mathbf{T}_2$. These functions are required to be bijective as we assume, that we want to depict the same situation with both views, i.e., sublanguages.

For coincident relation types, we presume an accordance of source and target object type $\mathbf{O}_s$ and $\mathbf{O}_t$. Therefore, besides a duplication of the relation type $\mathbf{R}$ we also request the assignment functions $F_s^{\mathbf{R}}$ and $F_t^{\mathbf{R}}$ to be duplicated: $F_s^{\mathbf{R}_1} : \mathbf{R}_1 \to \mathbf{O}_{s1}$ and $F_s^{\mathbf{R}_2} : \mathbf{R}_2 \to \mathbf{O}_{s2}$, thereby $\mathbf{O}_{s1}$ and $\mathbf{O}_{s2}$ being the renamed duplication of $\mathbf{O}_s$. Analogously we have to proceed with $F_t^{\mathbf{R}}$. We need constraints ensuring the coincidence of assigned values:

$$\forall r \in \mathbf{R}_1 \; i_{\mathbf{O}_{s1}}(F_s^{\mathbf{R}_1}(r)) = F_s^{\mathbf{R}_2}(i_{\mathbf{R}_1}(r)) \tag{92}$$

$$\forall r \in \mathbf{R}_1 \; i_{\mathbf{O}_{t1}}(F_t^{\mathbf{R}_1}(r)) = F_t^{\mathbf{R}_2}(i_{\mathbf{R}_1}(r)) \tag{93}$$

Besides assignment functions $F_s^{\mathbf{R}}$ and $F_t^{\mathbf{R}}$ we do not require accordance of attribute functions of object or relation types. This is to allow for the identification of same types appearing in different languages on different levels of detail. If coincident object or relation types also share attributes $\mathbf{A}$, this means $F^{\mathbf{A}} : \mathbf{T} \to \mathbf{T}'$ has coincident domain and codomain in both languages, we need an additional constraint to ensure the accordance of assigned attribute value:

$$\forall x \in \mathbf{T} \; i_{\mathbf{T}'}(f_1^a(x)) = f_2^a(i_{\mathbf{T}}(x)). \tag{94}$$

Coinciding data types $\mathbf{T}$ are also duplicated to $\mathbf{T}_1$ and $\mathbf{T}_2$ and connected via a bijection $i : \mathbf{T}_1 \to \mathbf{T}_2$. For product types $\mathbf{T} = \mathbf{T}_1 \times \mathbf{T}_2 \times \cdots \times \mathbf{T}_n$ coincidence means, that all participating types $\mathbf{T}_i$ coincide in both languages.

With this union of types the main part of the new language $\bar{\mathcal{L}}$ is fixed. Also, the specialization relations do not change and stay separated for both initial languages. All other signature elements not corresponding or belonging to an object type, relation type, or attribute, i.e., having no correspondence to a modeling concept, e.g., order relations, have to be considered situational from case to case. The constraints $\mathbf{C}$ are extended with the requirement that the mappings of the coinciding types $i : \mathbf{T}_1 \to \mathbf{T}_2$ are bijective, as well as the constraints on source and target assignment functions and attribute functions defined above. Existing constraints have to adopt the changes in symbol naming.

The power of language interleaving with formal languages lies in the ease of enriching the fusion of signatures with

added symbols and constraints tying the models together. To create new references between different views, we may introduce new attributes **A** with $\mathbf{F}^{\mathbf{A}} : \mathbf{T}_{dom} \rightarrow \mathbf{T}_{val}$, where attributed type $\mathbf{T}_{dom}$ and value domain $\mathbf{T}_{val}$ might stem from different initial languages. Newly defined constraints can be used to keep different views consistent. To define the attributes and constraints as required we might also have to introduce new product types in $\mathcal{S}_{\mathbf{D}}$ and additional function and relation symbols in $\mathcal{F}$ and $\mathcal{R}$, respectively.

If we allow views to be overlapping but not necessarily congruent, the mapping between two types $\mathbf{T}_1$ and $\mathbf{T}_2$ needs to be a partial function that is injective. This has several consequences for the newly defined signature, e.g., the constraint for coincident attributes has to be defined more carefully, as the mappings are not necessarily defined for each element.

This procedure of duplicating coinciding types, based on the ϵ-connection approach presented in [4] is promising because it allows for convenient handling of coincident types. By the duplication, we can prevent issues of divergent attributes of the same types and we have the possibility to resolve forced inconsistencies that might be induced by a join of coincident types. This is based on the full control over the degree of congruence of the types and other symbols by defining adequate constraints, e.g., for bijection, or less strictly defined for partial overlapping, for one-way attribute transfer, etc. Nevertheless, constraints on duplicated types can cause the set of valid models to shrink or even to cause unsatisfiability if the languages are per se not compatible.

5.2.3 Case study

We will demonstrate the procedure of interleaving on a case study of UML class diagrams and sequence diagrams. First, we have to formalize the initial languages $\mathcal{CD}$ of class diagrams and $\mathcal{SD}$ of sequence diagrams. For an easier comprehension, the considered signatures restrict to a subset of the original UML concepts relevant for the connection of the two languages, see Fig. 10.

Example 7 The UML class diagram language $\mathcal{CD}$
For our purpose, it suffices to consider in the signature $\Sigma_{\mathcal{CD}}$ only one object type **Class** (**Cl**) and one relation type **Association** (**As**) connecting classes. Besides the plain data types **Visibility** (**V**) and **SimpleType** (**ST**), class diagrams also provide the construct of attributes and operations of classes exhibiting a complex structure themselves. We define a data type called **Attribute** (**At**) which is a tuple of an element of type **Visibility** and the value of the attribute, i.e., an object of **ComplexType** (**CT**) = **SimpleType** ∪ **Class**. Furthermore we need a data type **Operation** (**Op**) which is a tuple of an element of type **Visibility**, a return type element in **CT**, and arbitrary many parameters of type **CT**. For these parameters, we use the union of all product types $(\mathbf{CT})^i$. For reasons of brevity we skip the explicit definition of all intermediate product types in the signature and just refer to $\bigcup_i (\mathbf{CT})^i$. As classes can have several (distinct) attributes and operations, the model attributes **C**(*lass*)**At**(*tributes*) and **C**(*lass*)**Op**(*erations*) point to the powersets of types $\wp(\mathbf{AT})$ and $\wp(\mathbf{OP})$.

The signature of $\mathcal{CD}$ looks as follows:

$$\Sigma_{\mathcal{CD}} = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}, \mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D \tag{95}$$
$$\mathcal{S}_O = \{\mathbf{Class}\}, \tag{96}$$
$$\mathcal{S}_R = \{\mathbf{Association}\}, \tag{97}$$
$$\begin{aligned} \mathcal{S}_D = \{&\mathbf{Visibility}, \mathbf{SimpleType}, \\ &\mathbf{ComplexType} = \mathbf{ST} \cup \mathbf{Cl}, \\ &\mathbf{Attribute} = \mathbf{V} \times \mathbf{CT}, \wp(\mathbf{At}), \\ &\mathbf{Operation} = \mathbf{V} \times \bigcup_i (\mathbf{CT})^i \times \mathbf{CT}, \wp(\mathbf{Op})\}, \end{aligned} \tag{98}$$
$$\begin{aligned} \mathcal{F} = \{&F_s^{\mathbf{As}} : \mathbf{As} \rightarrow \mathbf{Cl}, F_t^{\mathbf{As}} : \mathbf{As} \rightarrow \mathbf{Cl}, \\ &F^{\mathbf{CAt}} : \mathbf{Cl} \rightarrow \wp(\mathbf{At}),\ F^{\mathbf{COp}} : \mathbf{Cl} \rightarrow \wp(\mathbf{Op})\} \end{aligned} \tag{99}$$
$$\mathcal{R} = \{\}, \tag{100}$$
$$\mathcal{C} = \{+, -, \sim, String, Integer, Real, Boolean\} \tag{101}$$

Thereby, $+, -, \sim$ are of type **Visibility** and *String, Integer, Real*, and *Boolean* are of type **SimpleType**.

We do not need any postulates on the language $\mathcal{CD}$.

Example 8 The UML sequence diagram language $\mathcal{SD}$
Also in this example we restrict to the simplified case of having only one object type **Lifeline** (**Ll**) and two relation types **Message** (**Msg**) and **Replymessage** (**Rmsg**) both connecting lifelines. The temporal sequence of messages usually captured in the graphical order of arrows is defined in the attribute **Sendtime** (**MSt** and **RSt**) with value domain $\mathbb{N}_0$ assigning a point in time to the messages and replymessages. To be able to compare sendtimes we need the usual order relation $<_{time} \subseteq \mathbb{N}_0 \times \mathbb{N}_0$ and the usual addition function $+_{time}$in the signature:

$$\Sigma_{\mathcal{SD}} = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}, \mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D \tag{102}$$
$$\mathcal{S}_O = \{\mathbf{Lifeline}\}, \tag{103}$$
$$\mathcal{S}_R = \{\mathbf{Message}, \mathbf{Replymessage}\}, \tag{104}$$
$$\mathcal{S}_D = \{\mathbb{N}_0\}, \tag{105}$$
$$\begin{aligned} \mathcal{F} = \{&F^{\mathbf{MSt}} : \mathbf{Msg} \rightarrow \mathbb{N}_0, F^{\mathbf{RSt}} : \mathbf{Rmsg} \rightarrow \mathbb{N}_0, \\ &F_s^{\mathbf{Msg}} : \mathbf{Msg} \rightarrow \mathbf{Ll}, F_t^{\mathbf{Msg}} : \mathbf{Msg} \rightarrow \mathbf{Ll}, \\ &F_s^{\mathbf{Rmsg}} : \mathbf{Rmsg} \rightarrow \mathbf{Ll}, F_t^{\mathbf{Rmsg}} : \mathbf{Rmsg} \rightarrow \mathbf{Ll} \\ &+_{time} : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0\} \end{aligned} \tag{106}$$
$$\mathcal{R} = \{<_{time} \subseteq \mathbb{N}_0 \times \mathbb{N}_0\}, \tag{107}$$
$$\mathcal{C} = \{0, 1, 2, \ldots\} \text{ of type } \mathbb{N}_0 \tag{108}$$

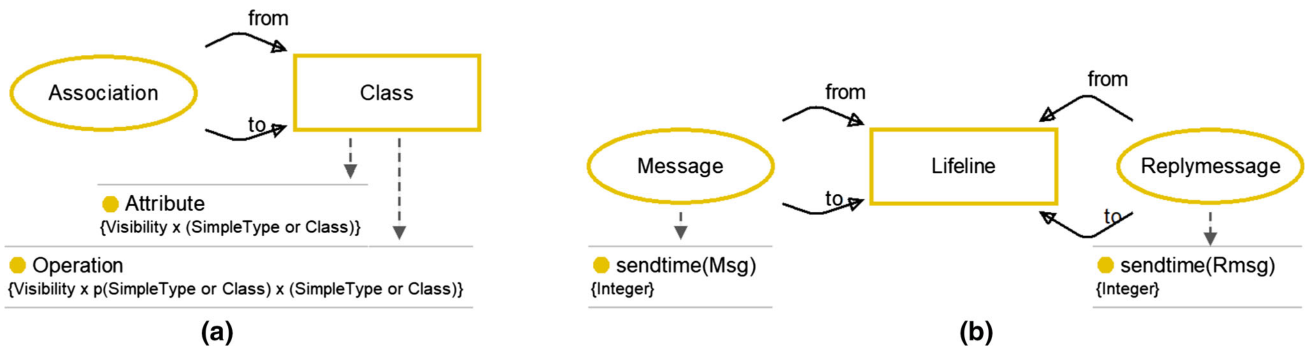

Fig. 10 Simplified metamodels of the UML class diagram (a) and sequence diagrams (b) (notation see Fig. 2)

To ensure a reasonable temporal flow of messages, we need two language constraints:

$$\forall x, y \in \mathbf{Msg}\ (F^{\mathbf{MSt}}(x) <_{time} F^{\mathbf{MSt}}(y) \vee F^{\mathbf{MSt}}(y) <_{time} F^{\mathbf{MSt}}(x)) \vee x = y \tag{109}$$

$$\forall x \in \mathbf{Rmsg}, \exists y \in \mathbf{Msg}\ (F^{\mathbf{MSt}}(y) +_{time} 1 = F^{\mathbf{RSt}}(x)) \tag{110}$$

Equation (109) restricts diagrams to be sequential, so no two messages are sent at the same time. Equation (110) forces the message flow to be synchronous.

Example 9 The interleaved modeling language $\mathcal{CD} \uplus \mathcal{SD}$
In the case of UML class diagrams and sequence diagrams we do not have to take care of identical types. We define several new attributes to bind lifelines in a sequence diagram to the classes in the corresponding class diagram: a reference **L**(*ife*)**l**(*ine*)**Cl**(*as*), a reference **Cal**(*led*)**Op**(*eration*) of a message, and a reference **Re**(*turn*)**Ty**(*pe*) of a replymessage.

$$F^{\mathbf{LlCl}} : \mathbf{Lifeline} \rightarrow \mathbf{Class} \tag{111}$$

$$F^{\mathbf{CalOp}} : \mathbf{Message} \rightarrow \mathbf{Operation} \tag{112}$$

$$F^{\mathbf{ReTy}} : \mathbf{Replymessage} \rightarrow \mathbf{ComplexType} \tag{113}$$

These references of course require some new constraints. To formulate these we need a new relation symbol $\in_{Op}$ between operations and sets thereof and a new function symbol F_{pr} projecting an element of type **Operation** to its returntype, i.e., the last value of the tuple.

$$\in_{\mathbf{Op}} \subseteq \mathbf{Operation} \times \wp(\mathbf{Operation}), \tag{114}$$

$$F_{pr} : \mathbf{Operation} \rightarrow \mathbf{ComplexType} \tag{115}$$

$$\forall x = (x_1, \ldots, x_n) \in \mathbf{Operation}\ F_{pr}(x) = x_n \tag{116}$$

With these symbols, we can define the additional constraints:

$$\forall x \in \mathbf{Msg}\ (F^{\mathbf{CalOp}}(x) \in_{\mathbf{Op}} F^{\mathbf{COp}}(F^{\mathbf{LlCl}}(F_t^{\mathbf{Msg}}(x)))) \tag{117}$$

$$\forall x \in \mathbf{Rmsg}\ \exists y \in \mathbf{Msg}(F^{\mathbf{MSt}}(y) +_{time} 1 = F^{\mathbf{RSt}}(x) \wedge F^{\mathbf{ReTy}}(x) = F_{pr}(F^{\mathbf{CalOp}}(y))) \tag{118}$$

Equation (117) ensures that a message can only call operations of the addressed class. Equation (118) guarantees that each replymessage follows a message and the returntype is exactly the returntype of the called operation.

The complete language $\mathcal{CD} \uplus \mathcal{SD}$ looks as follows:

$$\Sigma_{\mathcal{CD}\uplus\mathcal{SD}} = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}, \mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D \tag{119}$$

$$\mathcal{S}_O = \{\mathbf{Class}, \mathbf{Lifeline}\}, \tag{120}$$

$$\mathcal{S}_R = \{\mathbf{Association}, \mathbf{Message}, \mathbf{Replymessage}\}, \tag{121}$$

$$\mathcal{S}_D = \{\mathbf{Visibility}, \mathbf{SimpleType}, \mathbf{ComplexType}, \mathbf{Attribute}, \mathbf{Operation}, \wp(\mathbf{At}), \wp(\mathbf{Op}), \mathbb{N}_0\} \tag{122}$$

$$\begin{aligned}\mathcal{F} = \{&F_s^{\mathbf{As}} : \mathbf{As} \rightarrow \mathbf{Cl}, F_t^{\mathbf{As}} : \mathbf{As} \rightarrow \mathbf{Cl},\\ &F^{\mathbf{CAt}} : \mathbf{Cl} \rightarrow \wp(\mathbf{At}), F^{\mathbf{COp}} : \mathbf{Cl} \rightarrow \wp(\mathbf{Op}),\\ &F_s^{\mathbf{Msg}} : \mathbf{Msg} \rightarrow \mathbf{Ll}, F_t^{\mathbf{Msg}} : \mathbf{Msg} \rightarrow \mathbf{Ll},\\ &F_s^{\mathbf{Rmsg}} : \mathbf{Rmsg} \rightarrow \mathbf{Ll}, F_t^{\mathbf{Rmsg}} : \mathbf{Rmsg} \rightarrow \mathbf{Ll},\\ &F^{\mathbf{MSt}} : \mathbf{Msg} \rightarrow \mathbb{N}_0, F^{\mathbf{RSt}} : \mathbf{Rmsg} \rightarrow \mathbb{N}_0,\\ &F^{\mathbf{LlCl}} : \mathbf{Lifeline} \rightarrow \mathbf{Class}\\ &F^{\mathbf{CalOp}} : \mathbf{Message} \rightarrow \mathbf{Operation}\\ &F^{\mathbf{ReTy}} : \mathbf{Replymessage} \rightarrow \mathbf{ComplexType}\\ &F_{pr} : \mathbf{Operation} \rightarrow \mathbf{ComplexType}\\ &+_{time} : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0\}\end{aligned} \tag{123}$$

$$\mathcal{R} = \{<_{time} \subseteq \mathbb{N}_0 \times \mathbb{N}_0, \in_{\mathbf{Op}} \subseteq \mathbf{Op} \times \wp(\mathbf{Op})\} \tag{124}$$

$$\mathcal{C} = \{+, -, \sim, String, Integer, Real, Boolean, 0, 1, 2, \ldots\} \tag{125}$$

With the newly generated language each model contains all information of both views, the structural view of class diagrams as well as the procedural view of sequence diagrams.

Of course, when viewing the model we only consider the model restricted to a sublanguage, $\mathcal{CD}$ or $\mathcal{SD}$, but in the background all elements of both reside in the "supermodel." This means all information is captured in the model at all points in time and at the same time kept consistent due to the newly introduced constraints. This conforms to the idea of the single underlying model as proposed by Burger et al. [49].

5.3 Operations on models

Model functionality is a crucial point to amplify the value of models beyond mere pictures [6]. One of the most prominent examples is the firing mechanism on Petri Nets [58]. Also, many domain-specific languages gain in value by the offered model operations. For example, model operations in the sense of model to model transformations play a crucial role in model-driven software engineering [11, Chap. 8]. Nevertheless, operations on models are often out of scope or simply ignored in formalizations. An exception is the theory of graph grammars and graph transformations [30]. This approach is based on the principle of finding and replacing patterns in labeled graphs with other patterns. As we chose a fundamentally different underlying structural theory conforming to the linguistic character of modeling languages, we approach operations on models using a notion of models beyond box and line constructs. This shortcoming of graph grammars manifests itself in the lack of a canonical correspondent in graphs for model attributes and their change operations.

Formalisms based on logic are often critiqued for not being able to capture the operational syntax of modeling languages. We argue that this is not an inevitable inability of these approaches and show some ideas on how operations on models can also be supported by concepts from logic.

5.3.1 Structural events and domain events

We adopt the notion of Olivé [53, Chap. 11] who defines domain events, i.e., semantically and syntactically admissible operations on models, by decomposing them into the smallest possible changes in a model, the so called structural events. Domain events therefore always map valid models to other valid models conforming to the language postulates, while structural events can invalidate a model and cause a violation of the postulates.

While Olivé only names deletion and insertion of objects and relations as structural events, for our purpose in the MetaMoprh formalism we also have to consider the change of attribute values as a third variant.

Each of these events requires a closer look at the preconditions and consequences. A *create* event of an object type $\mathbf{T}$ implies the need to set the values of all attributes of the concrete object to an initial value, e.g., if we create a place in a Petri Net, we have to initially set the *tokens* attribute, e.g., to 0. When deleting a concrete object obj of type $\mathbf{T}$ we also have to delete the relations that start or end at obj and we have to reset all attributes of any other element pointing to obj. This shows that already for compliance with the foundational definitions structural events often occur in groups.

Consider again model $\mathtt{M}$ of Example 2 of the Petri Net model depicting a barber shop. A valid domain event is the firing of the transition *Serve*. This event is the concatenation of the three structural events of changing the attribute values: first the value of the *tokens* attribute of element *Busy* is set to 1, the *tokens* attribute of *Idle* is set to 0, and the *tokens* attribute of *Wait* is set to 1. None of these structural events alone is semantically valid, but together they form a semantically and syntactically admissible operation. This also shows that there are many structural events (we can set the attribute *tokens* of each place to any number we like) but much less domain events.

Concatenations of domain events form sequences of valid models

$$\mathtt{M}_0 \mapsto \mathtt{M}_1 \mapsto \mathtt{M}_2 \mapsto \cdots .$$

In Petri Nets, for example, the firing of a transition is a domain event. Therefore these sequences are of special interest as they reveal inaccessible states and final markings in a net when starting from a concrete model. This is closely related to the concept of marking graphs in Petri Nets [58, Sec. 2.8].

Another point to be considered are pre- and postconditions of domain events. To capture these in a generic way we can use concepts from temporal logic [42]. With the logical operators from temporal logic we are able to formulate postulates considering both states of a model, before and after the application of a domain event, and define dependencies between both.

With a formalism that is means to capture transformation of models—be it in imperative style with structural and domain events or in a declarative style with pre- and postconditions or both mixed—mechanisms and algorithms become an integral part of the specification of a modeling language. Shifting the ascertainment of functionality from the implementation level to the design level this functionality can be specified uniquely and rigorously and becomes independent from the concrete software in use.

5.4 Translators

Another salient benefit of having an unambiguous and complete formalization of a modeling language is that it can serve as a single point of platform-independent specification, thereby being precise enough to be automatically processed by a machine. Of course, a modeling language without a tech-

nical tool supporting the creation and execution of models is very much useless for the target audience. When implementing a language many engineers have made the experience that available metamodeling platforms differ heavily in available concepts and functionality and thereby impose more or less severe restrictions on the final product [38]. So the implementation forces the engineer to think in the frame of the used platform and to modify the language to fit the given meta^{2}model and available model processing algorithms. A further drawback of this current practice is the fact that each effort of implementation is lost whenever the language has to be transferred to another platform, may it be caused by missing functionality for new language features or cessation of platform support.

With the formalization of a language as stipulated by the AMME lifecycle of modeling methods, we derive a sort of platform-independent code and close the gap between the specification document and the final implementation. By using the proposed formalism the specification of the main concepts is unified and therefore offers the possibility to be translated to any metamodeling platform. Thus, the language specification stays on a platform-agnostic level and the complexity of the platform-specificity can be outsourced to a platform-specific translator. The feasibility of this endeavor has been shown by Visic et al. [64]. When platforms change, only the translator has to be adapted but not the platform-independent conceptualization of a language.

While Visic et al. stay at the level of translators of language syntax, our attempt on the formalization of model operations shown in Sect. 5.3 holds promise to be able to integrate an automatic translation of the functionality of modeling languages. The decomposition of domain events into the three types of structural events allows for an automatization of translating the modeling language specification to a concrete tool as most platforms offer methods for creating or deleting elements or changing attributes.

6 Discussion

The conceptual modeling formalism `MetaMorph` allows for a precise definition of modeling languages. The formalization of a modeling language with `MetaMorph` requires a full declaration of any element possibly instantiated, i.e., any symbol in the signature of the language. The formalization process obliges the engineer to make explicit any concept and possible instances or values of the language as well as to unfold any constraint to prevent not permitted model constructs. Therefore, it results in a complete specification ready for use.

The constraints formulated in first-order logic allow for checking the correctness of a concrete model. For automation, also a model-checking software can be employed. Correctness of the specification, on the other hand, can be checked with model-finding software. These tools can help to determine if the language is underconstrained or overconstrained by delivering unintended models or not delivering intended models, respectively. Also, inconsistent specifications, i.e., specifications with unsatisfiable postulates, can be detected with model-finders. The choice of domain concepts is the task of the language engineer, and its correctness can of course only be checked by a thorough evaluation by a domain expert but not by a formalism.

To evaluate the MetaMorph formalism we recap the requirements mentioned in Sect. 1.2: 1) The formalism has to be complete regarding the general building blocks of a language, 2) it must comply with the linguistic character of modeling languages, 3) it must be generic in a way that it admits the formalization of any language developed according to the four-layer metamodeling stack, and 4) it must provide an integrative formal foundation offering canonical tools for the advancements in conceptual modeling research.

The proposed formalism comprises the core concepts constituting a modeling language. These are chosen based on a survey by Kern et al. [38] and the concept discussion by Olivé [53]. We restrict to the most common concepts, i.e., those appearing in at least half of the surveyed metamodeling platforms in [38]. In Sect. 3, we also list the concepts for future integration. Regarding the first requirement, we conclude that the proposed definition of a modeling language is not yet complete but depicts the most relevant core. This is also shown by the realizability of several case studies depicted in this paper.

In current research, the notion of formal languages in the sense of mathematical logic as underlying structure for modeling languages has been receiving increasing attention [18,28,53,56,63]. This supports our choice of using logic as basis for the formalism and underpins the adherence to the linguistic character of languages including the alphabet and the instantiation relation. Additional affirmation is given by the multitude of practical constructs and methods of formal language theory and its straightforward applicability to current research issues, which is exemplarily shown in Sect. 5.

The requirement for generic realizability of arbitrary modeling languages developed according to the four-layer metamodeling stack is satisfied, as, by construction, MetaMorph allows for the definition of exactly those concepts constituting a language that are core to conceptual modeling according to our literature review. A realization of the four divergent use cases in this paper and several more use cases conducted by the author furthermore backs this claim.

The integrability of the MetaMorph formalism is outlined in Sect. 5. There, we approach four diverse research topics ranging from the integration of the advanced concept of power types, over the interleaving of modeling languages, to the formal definition of model transformation. For all these

topics, the formalism has canonical tools at hand. This is not proof but a strong indication for MetaMorph being an eligible formal foundation for conceptual modeling.

The empirical evaluation of feasibility and usability so far has been mainly conducted via the realization of prototypical case studies of various domains. Four of them are shown in this paper. Other cases guiding the advancement of the formalism are, for example, ER-diagrams starting in [20]. In the light of language interleaving we formalized a (yet unpublished) language for modeling smart cities [8] besides the UML case study. To investigate the formalization of model operations we formalized Petri Nets and ProVis, a tool for math education providing sophisticated methods to process statistical diagrams [22].

In parallel, a more outreaching empirical evaluation is currently being conducted. Students in the business informatics program were asked to participate in the evaluation by attending an introductory lecture about `MetaMorph` and to formalize their own metamodeling projects implemented for the course *Metamodeling* part of the masters program. After the formalization they filled in a questionnaire about adequacy and usability of `MetaMorph` as well as their pre-knowledge in modeling, metamodeling, and logic. The goal of the evaluation is to measure the actual intuitivity of the formalism and satisfaction of language engineers. We are also interested in the influence of pre-knowledge on the use of the proposed formalism. Further questions to be answered are how much effort it takes to formalize a small-sized language, if the size and complexity of the language have an influence on the experienced complexity when using the formalism and if an introductory session is helpful or even required to successfully apply MetaMorph. The evaluation of the questionnaires considering the actual outcome of the students' formalizations (correctness, extent, …) is currently in progress.

A proof of concept for the significance of the presented formalism can be given by an implementation of translators to at least two different metamodeling platforms, especially if we are able to integrate a specification of model operations. Such a tool is currently under design.

7 Conclusion

In this paper, we presented the MetaMorph formalism comprising a definition of modeling languages as formal languages $\mathcal{L}$ with a signature Σ in the sense of logic. The concept of a $\mathcal{L}$-structure canonically corresponds to a model being expressed with a modeling language and led us to the definition of models as $\mathcal{L}$-structures. To illustrate the specification of formal modeling languages we demonstrated the definition on the Petri Nets modeling language. We applied the definition also on the meta-level and developed `M2FOL`—a formal modeling language for metamodels. `M2FOL` models are precise and complete and therefore we were able to show how to algorithmically derive a formal modeling language signature from its metamodel. `M2FOL` is self-describing, which can be seen by applying the algorithm to its own metamodel.

After the introduction of the formalism, we gave an outlook to the potential and benefits of formalized modeling languages using the approach at hand. We outlined the ease of extensibility of the formalism by integrating the concept of power types, thereby also building a bridge to multi-level modeling approaches. To demonstrate the use of MetaMorph with power types, we realized a case study on a customizable process modeling language.

Furthermore, we addressed the topic of language interleaving and consistency. Established methods from formal language theory provide methods to create an interleaved formal language from existing ones. We illustrated the process on a case study using UML class diagrams and sequence diagrams.

Another topic with high potential for the automatization of language implementation is the formalization of model operations. We outlined how to break down algorithms on models in the smallest possible building blocks able to be formalized. This allows model operations to become an integral part of the formal language specification.

This formal specification—syntax as well as operations—precise enough to be processed by a machine yet platform-independent additionally allows us to develop platform-specific translators, transferring the single source of language specification to realizations on different platforms.

With this common approach to defining metamodels and modeling languages, these languages become comparable, reusable, and open to modularization. To broaden the conceptual capabilities of our approach, we will further investigate more subtle concepts to be integrated into the definition. These are, for example, the concepts of mixins and extenders for modular metamodels as proposed in [65], or the further structural relations identified in [47] besides the *intension/extension* relation that was already realized with power types. For a practical application of the language `M2FOL`, a suitable tool for transforming graphical metamodels into formal ones will be developed.

Finally, by using a sophisticated mathematical theory as grounding for the definition of modeling languages we can use this knowledge stack as a resource to further establish a formal foundation for modeling languages. We can investigate the subclass of conceptual modeling languages in the class of formal languages and approach old problems with new tools.

Funding Open access funding provided by University of Vienna.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Almeida, J.P.A., Musso, F.A., Carvalho, V.A., Fonseca, C.M., Guizzardi, G.: Capturing multi-level models in a two-level formal modeling technique. In: Laender, A., Pernici, B., Lim, E.P., de Oliveira, J. (eds.) Conceptual Modeling. ER 2019. Lecture Notes in Computer Science, vol. 11788 LNCS, pp. 43–51. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33223-5_5
2. Awadid, A., Nurcan, S.: Consistency requirements in business process modeling: a thorough overview. Softw. Syst. Model. **18**(2), 1097–1115 (2019). https://doi.org/10.1007/s10270-017-0629-2
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2007). https://doi.org/10.1017/CBO9780511711787
4. Baader, F., Ghilardi, S.: Connecting many-sorted theories. J. Symb. Log. **72**(2), 535–583 (2007). https://doi.org/10.2178/jsl/1185803623
5. BOC: ADOxx Metamodelling Platform (2021). https://www.adoxx.org
6. Bork, D., Buchmann, R.A., Karagiannis, D., Lee, M., Miron, E.T.: An open platform for modeling method conceptualization: the OMiLAB digital ecosystem. Commun. Assoc. Inf. Syst. **44**(1), 673–679 (2019). https://doi.org/10.17705/1CAIS.04432
7. Bork, D., Fill, H.G.: Formal aspects of enterprise modeling methods: a comparison framework. In: 47th Hawaii International Conference on System Sciences, pp. 3400–3409 (2014). https://doi.org/10.1109/HICSS.2014.422
8. Bork, D., Fill, H.G., Karagiannis, D., Miron, E.T., Tantouris, N., Walch, M.: Conceptual modelling for smart cities: a teaching case. IxD&A, *27*, 10–27 (2015)
9. Bork, D., Karagiannis, D., Pittl, B.: A survey of modeling language specification techniques. Inf. Syst. **87**, 101425 (2020). https://doi.org/10.1016/j.is.2019.101425
10. Brachman, R.J., Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. In: Mylopolous, J., Brodie, M. (eds.) Readings in Artificial Intelligence and Databases, pp. 207–230. Morgan Kaufmann, San Francisco, CA (1989)
11. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice: second edition. Synth. Lect. Softw. Eng. **3**(1), 1–207 (2017). https://doi.org/10.2200/s00751ed2v01y201701swe004
12. Buchmann, R.A., Ghiran, A.M., Döller, V., Karagiannis, D.: Conceptual modeling education as a "design problem". Complex Syst. Inform. Model. Q. **21**, 21–33 (2019). https://doi.org/10.7250/csimq.2019-21.02
13. Burger, E.: Flexible Views for View-Based Model-driven Development. KIT Scientific Publishing (2014). https://doi.org/10.5445/KSP/1000043437
14. Cadavid, J., Combemale, B., Baudry, B.: Ten years of meta-object facility: an analysis of metamodeling practices. Research Report RR-7882, INRIA (2012). https://hal.inria.fr/hal-00670652
15. Carvalho, V.A., Almeida, J.P.A.: Toward a well-founded theory for multi-level conceptual modeling. Softw. Syst. Model. **17**(1), 205–231 (2018). https://doi.org/10.1007/s10270-016-0538-9
16. Choe, Y., Lee, S., Lee, M.: SAVE: an environment for visual specification and verification of IoT. In: IEEE 20th International Enterprise Distributed Object Computing Workshop, EDOCW, pp. 269–276 (2016). https://doi.org/10.1109/EDOCW.2016.7584384
17. Clyde, S.W., Embley, D.W., Liddle, S.W., Woodfield, S.N.: OSM-logic: a fact-oriented, time-dependent formalization of object-oriented systems modeling. In: Conceptual Modelling and Its Theoretical Foundations, pp. 151–172. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-28279-9_12
18. Delcambre, L.M.L., Liddle, S.W., Pastor, O., Storey, V.C.: A reference framework for conceptual modeling. In: Trujillo, J.C., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (eds.) Conceptual Modeling. ER 2018. Lecture Notes in Computer Science, vol. 11157 LNCS, pp. 27–42. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00847-5_4
19. Delcambre, L.M.L., Liddle, S.W., Pastor, O., Storey, V.C.: Characterizing conceptual modeling research. In: Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R. (eds.) On the Move to Meaningful Internet Systems: OTM 2019 Conferences, pp. 40–57. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33246-4_3
20. Döller, V.: 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, vol. 2234, pp. 61–73. CEUR-WS (2018)
21. Döller, V.: M2FOL: a formal modeling language for metamodels. In: Bork, D., Grabis, J. (eds.) The Practice of Enterprise Modeling. PoEM 2020. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-63479-7_8
22. Döller, V., Karagiannis, D.: Formalizing conceptual modeling methods with MetaMorph. In: Augusto, A., Gill, A., Nurcan, S., Reinhartz-Berger, I., Schmidt, R., Zdravkovic, J. (eds.) Enterprise, Business-Process and Information Systems Modeling. BPMDS 2021, EMMSAD 2021. Lecture Notes in Business Information Processing, pp. 245–261. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79186-5_16
23. Enderton, H.B.: A Mathematical Introduction To Logic, 2nd edn. Harcourt/Academic Press, San Diego (2001)
24. Fill, H.G., Redmond, T., Karagiannis, D.: FDMM: a formalism for describing ADOxx meta models and models. In: ICEIS 2012—Proceedings of the 14th International Conference on Enterprise Information Systems, vol. 3, pp. 133–144 (2012). https://doi.org/10.5220/0003971201330144
25. Frank, U.: Domain-specific modeling languages: requirements analysis and design guidelines. In: Reinhartz-Berger, I., Sturm, A., Clark, T., Cohen, S., Bettin, J. (eds.) Domain Engineering: Product Lines, Languages, and Conceptual Models, pp. 133–157. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-36654-3_6
26. Frank, U.: Multilevel modeling: toward a new paradigm of conceptual modeling and information systems design. Bus. Inf. Syst. Eng. **6**, 319–337 (2014). https://doi.org/10.1007/s12599-014-0350-4
27. Frank, U., Strecker, S., Fettke, P., Vom Brocke, J., Becker, J., Sinz, E.: The research field "modeling business information systems": current challenges and elements of a future research agenda. Bus. Inf. Syst. Eng. **6**, 39–43 (2014). https://doi.org/10.1007/s12599-013-0301-5
28. Guarino, N., Guizzardi, G., Mylopoulos, J.: On the philosophical foundations of conceptual models. In: Proceedings of the 29th International Conference on Information Modelling and Knowl-

edge Bases, EJC 2019. Frontiers in Artificial Intelligence and Applications, vol. 321, pp. 1–15. IOS Press (2019). https://doi.org/10.3233/FAIA200002
29. Guizzardi, G.: On ontology, ontologies, conceptualizations, modeling languages, and (meta)models. In: Vasilecas, O., Eder, J., Caplinskas, A. (eds.) Selected Papers from the Seventh International Baltic Conference, DB&IS. pp. 18–39. IOS Press (2007)
30. Heckel, R., Taentzer, G.: Graph Transformation for Software Engineers. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-43916-3
31. Henderson-Sellers, B.: On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages. Springer, Berlin (2012)
32. Herrmann, C., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: An algebraic view on the semantics of model composition. In: Model Driven Architecture-Foundations and Applications (ECMDA-FA), pp. 99–113. Springer (2007). https://doi.org/10.1007/978-3-540-72901-3_8
33. Jackson, E., Sztipanovits, J.: Formalizing the structural semantics of domain-specific modeling languages. Softw. Syst. Model. **8**(4), 451–478 (2009). https://doi.org/10.1007/s10270-008-0105-0
34. Jouault, F., Bézivin, J.: KM3: a DSL for metamodel specification. In: International Conference on Formal Methods for Open Object-Based Distributed Systems, FMOODS 2006, pp. 171–185. Springer, Berlin (2006). https://doi.org/10.1007/11768869_14
35. Kampars, J., Zdravkovic, J., Stirna, J., Grabis, J.: Extending organizational capabilities with open data to support sustainable and dynamic business ecosystems. Softw. Syst. Model. **19**(2), 371–398 (2020). https://doi.org/10.1007/s10270-019-00756-7
36. Karagiannis, D.: Conceptual modelling methods: the AMME agile engineering approach. In: Informatics in Economy. IE 2016. Lecture Notes in Business Information Processing, vol. 273, pp. 3–19. Springer (2018). https://doi.org/10.1007/978-3-319-73459-0_1
37. Karagiannis, D., Burzynski, P., Utz, W., Buchmann, R.A.: A metamodeling approach to support the engineering of modeling method requirements. In: 27th IEEE International Requirements Engineering Conference, pp. 199–210 (2019). https://doi.org/10.1109/RE.2019.00030
38. Kern, H., Hummel, A., Kühne, S.: 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, pp. 7–12. ACM (2011). https://doi.org/10.1145/2095050.2095053
39. Klare, H., Kramer, M.E., Langhammer, M., Werle, D., Burger, E., Reussner, R.: Enabling consistency in view-based system development—the VITRUVIUS approach. J. Syst. Softw. **171**, 110815 (2021). https://doi.org/10.1016/j.jss.2020.110815
40. Koubarakis, M., Borgida, A., Constantopoulos, P., Doerr, M., Jarke, M., Jeusfeld, M.A., Mylopoulos, J., Plexousakis, D.: A retrospective on Telos as a metamodeling language for requirements engineering. Requir. Eng. (2020). https://doi.org/10.1007/s00766-020-00329-x
41. Koubarakis, M., Mylopoulos, J., Stanley, M., Borgida, A.: Telos: features and formalization. Technical report KRRTR- 89-4, Department of Computer Science, University of Toronto (1989)
42. Kröger, F.: Temporal Logic and State Systems, 1st edn. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-68635-4
43. Kühne, T.: Matters of (meta-) modeling. Softw. Syst. Model. **5**(4), 369–385 (2006). https://doi.org/10.1007/s10270-006-0017-9
44. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: ϵ-connections of abstract description systems. Artif. Intell. **156**(1), 1–73 (2004). https://doi.org/10.1016/j.artint.2004.02.002
45. Lara, P., Sánchez, M., Villalobos, J.: Bridging the IT and OT Worlds using an extensible modeling language. In: Conceptual Modeling. ER 2016. Lecture Notes in Computer Science, vol. 9974, pp. 122–129. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_10
46. Leroy, D., Bousse, E., Wimmer, M., Mayerhofer, T., Combemale, B., Schwinger, W.: Behavioral interfaces for executable DSLs. Softw. Syst. Model. **19**(4), 1015–1043 (2020). https://doi.org/10.1007/s10270-020-00798-2
47. Mayr, H.C., Thalheim, B.: The triptych of conceptual modeling. Softw. Syst. Model. (2020). https://doi.org/10.1007/S10270-020-00836-Z
48. Mazzola, G., Milmeister, G., Weissmann, J.: Comprehensive Mathematics for Computer Scientists 1: Sets and Numbers, Graphs and Algebra, Logic and Machines, Linear Geometry, 2nd edn. Springer, Berlin Heidelberg (2006)
49. Meier, J., Klare, H., Tunjic, C., Atkinson, C., Burger, E., Reussner, R., Winter, A.: Single underlying models for projectional, multiview environments. In: MODELSWARD 2019—Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, pp. 119–130. SciTePress (2019). https://doi.org/10.5220/0007396401190130
50. Mouratidis, H., Giorgini, P.: Secure Tropos: a security-oriented extension of the Tropos methodology. Int. J. Softw. Eng. Knowl. Eng. **17**, 285–309 (2007). https://doi.org/10.1142/S0218194007003240
51. Object Management Group (OMG): Object Constraint Language—Version 2.4. (2014). https://www.omg.org/spec/OCL/2.4/PDF
52. Odell, J.: Power types. J. Object Oriented Program. **7**(2), 8–12 (1994)
53. Olivé, A.: Conceptual Modeling of Information Systems. Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-39390-0
54. O'Regan, G.: Z formal specification language. In: Concise Guide to Formal Methods: Theory, Fundamentals and Industry Applications, pp. 155–171. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64021-1_8
55. Paczona, M., Mayr, H.C.: Model-driven mechatronic system development. In: IEEE 15th International Conference on Automation Science and Engineering (CASE), pp. 1730–1736. IEEE (2019). https://doi.org/10.1109/COASE.2019.8843314
56. Partridge, C., Gonzalez-Perez, C., Henderson-Sellers, B.: Are conceptual models concept models? In: Conceptual Modeling. ER 2013. Lecture Notes in Computer Science, vol. 8217, pp. 96–105. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-41924-9_9
57. Ralyté, J., Léonard, M.: Evolution models for information systems evolution steering. In: Poels, G., Gailly, F., Serral Asensio, E., Snoeck, M. (eds.) The Practice of Enterprise Modeling, PoEM 2017, pp. 221–235. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70241-4_15
58. Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-33278-4
59. Sandkuhl, K., Fill, H.G., Hoppenbrouwers, S., Krogstie, J., Leue, A., Matthes, F., Opdahl, A.L., Schwabe, G., Uludag, Ö., Winter, R.: Enterprise modelling for the masses–from elitist discipline to common practice. In: Horkoff, J., Jeusfeld, M., Persson, A. (eds.) The Practice of Enterprise Modeling, PoEM 2016, vol. 267, pp. 225–240. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-48393-1_16
60. Schön, H., Zdravkovic, J., Stirna, J., Strahringer, S.: A role-based capability modeling approach for adaptive information systems. In: Gordijn, J., Guédria, W., Proper, H. (eds.) The Practice of Enterprise Modeling, PoEM 2019, vol. 369, pp. 68–82. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-35151-9_5
61. Spivey, J.M., Abrial, J.R.: The Z Notation: A Reference Manual, 2nd edn. Prentice-Hall, Hemel Hempstead (1992)

62. Stirna, J., Zdravkovic, J., Grabis, J., Sandkuhl, K.: Development of capability driven development methodology: experiences and recommendations. In: Poels, G., Gailly, F., Serral Asensio, E., Snoeck, M. (eds.) The Practice of Enterprise Modeling, PoEM 2017, pp. 251–266. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70241-4_17
63. Thalheim, B.: The theory of conceptual models, the theory of conceptual modelling and foundations of conceptual modelling. In: Handbook of Conceptual Modeling, pp. 543–577. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-15865-0_17
64. Visic, N.: Language-Oriented Modeling Method Engineering. Ph.D. thesis, University of Vienna (2016)
65. Zivkovic, S., Karagiannis, D.: Mixins and extenders for modular metamodel customisation. In: Proceedings of the 18th International Conference on Enterprise Information Systems, pp. 259–270. Science and and Technology Publications (2016). https://doi.org/10.5220/0005761102590270

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Victoria Döller is a research associate and Ph.D. candidate at the Research Group Knowledge Engineering at the Faculty of Computer Science, University of Vienna, Austria. She holds Diplomas (master's equivalent) in pure mathematics and in mathematics education from the University of Vienna and has professional experience as a software developer. Her research interests include domain-specific conceptual modeling, metamodeling, as well as formalizations and foundations of conceptual modeling.