M2FOL: A Formal Modeling Language for Metamodels

Victoria Döller

M2FOL: A Formal Modeling Language for Metamodels

Victoria Döller[0000−0002−5770−635X]

Research Group Knowledge Engineering,
Faculty of Computer Science,
University of Vienna, Vienna, Austria
victoria.doeller@univie.ac.at

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. In this paper we build on the widely accepted approach of logic as basis for modeling languages and define them as languages in the sense of typed predicate logic comprising a signature Σ and a set of constraints. We concretize how the basic concepts of a language – object and relation types, attributes, inheritance and constraints – can be expressed in logical terms. This naturally leads to the denotation of a model as Σ-structure satisfying all constraints. We apply this definition also on the metalevel and propose a formal modeling language to specify metamodels called `M2FOL`. A thus formalized metamodel then rigorously defines the signature of a language and we provide an algorithmic derivation of the formal modeling language from the metamodel. The effectiveness of our approach is demonstrated by formalizing the Petri Net modeling language, a method frequently used for analysis and simulation in enterprise modeling.

Keywords: Conceptual Modeling, Metamodel, 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 enhanced modeling languages from mere instruments for pictures supporting human understanding to highly specialized tools with value adding mechanisms like information querying, simulation, and transformation [2] [14]. The nature of models has evolved from a visual representation of information to an exploitable knowledge structure [5]. 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 [34] and [35] a research agenda is formulated to establish "modeling for the masses" (MftM) and broadcast its benefits also to non-experts.

Although the initiators of the MftM movement mention that the formality of model representation possibly hampers understandability, we argue that the idea behind MftM nevertheless requires an investigation of the formal foundations of models and languages. This is for three reasons: 1) According to the *stakeholder dimension* of challenges in enterprise modeling research [34, S.234] computers also have to be seen as stakeholders producing and consuming models. To make models computer-processable they have to be formalized as computers do not understand semi-formal or unstructured models and language specifications [3]. 2) The vision of models being not autotelic but *being a means to the operationalization of information* [34, S.229] calls for value-adding functionality beyond mere graphics like reasoning, verification & validation or simulation, which is formulated ideally computer-understandably and implementation-independently, i.e. formalized. 3) The vision of *local modeling practices which are globally integrative* [34, S.229] calls for a common foundation of what models and modeling languages are to enable the linking and merging of models in different domains with different semantics [18].

Formalization is also essential in the light of the emergent importance of domain specific modeling languages (DSMLs) [13] as well as an increasing agility in the advancement and extension of established languages and methods [21]. The lack of a common way for formalizing DSMLs leads to divergent 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 [28] [33]. 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 this impedes a mutual interconnection and integration by divergent base concepts of models and different underlying formal structures.

Another important building block for advancing the science of conceptual modeling is an exact and commonly applied method for specifying 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 [4]. 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. [32] [36] [37]. 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. [26] [30]. 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 [3].

In this paper, we contribute to fostering the formal foundations of modeling languages by defining them 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. Our definition concretely states how the core concepts of modeling languages as described in [23] and 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 being an instantiation of a metamodel. With this definition of a language we are then able to specify `M2FOL`, a formal modeling language for metamodels. With `M2FOL` we are capable of modeling the syntax of a language to be specified, particularly the signature of the language according to the definition.

The rest of this paper is structured as follows: In Section 2 we give an overview of related work on formalization of metamodels and modeling languages. In Section 3 we introduce our definition of formal modeling languages and models and concretize how the basic concepts of a language – object and relation types, attributes, inheritance and constraints – can be expressed in logical terms. We then use this definition in Section 4 to create `M2FOL` – a formal modeling language for metamodels – and outline its self-descriptive character. Given a metamodel specified with `M2FOL` we show how to algorithmically deduce the signature of the corresponding modeling language. Finally we give a conclusion and outlook to issues we plan to approach in the future with this formalization.

2 Background and Related Work

According to the Characterizing Conceptual Model Research (CCMR) framework we are interested in contributions located in the dimension *Formalize* working on the level of *Conceptual Modeling Languages* and *Metamodeling Languages* [9]. The various attempts addressing the formalization of a specific modeling language 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 behaviour [7]. Another example is the SAVE method for simulating IoT systems using the δ-calculus [6]. These specific formalizations may offer ideas suitable to be 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.

Research contributions on generic formal foundations of modeling languages can be categorized according to the underlying mathematical theory they use, mostly graph theory, set theory and logic.

In the domain specific language KM3 presented by Jouault and Bezivin models are defined as directed multi-graphs conforming to another model, the

metamodel, itself a graph [20]. Using this formalism the authors define a self-descriptive 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.

The FDMM formalism introduced by Fill et al. uses set theory to specify metamodels and models [12]. The authors explicitly aim at an formalization of metamodels realized with the metamodeling platform ADOxx [1] and do not claim to be applicable for platform-independent specifications. Neither set theory, basis of FDMM, nor graph theory, basis of KM3, provides a canonical concept for instantiation, an essential characteristic of modeling languages. Therefore the technique and semantics of this instantiation relation between model and metamodel has to be constructed ad-hoc and lacks the beneficial knowledge stack of established mathematical theories.

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 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 [8] [15] [29] [31] [40]. In his work on the theory of conceptual models Thalheim describes modeling languages as based on a signature Σ comprising a set of postulates, i.e. sentences expressed with elements of Σ [40]. Models are defined as language structures satisfying the postulates, which canonically corresponds to the concept of instantiation of a metamodel. This generic description does not restrict itself to a single type of logic. Consequently it does not provide a specification of the mapping of the formal signature to the modeling language concepts like object or relation types.

In their investigation of formal foundations of domain-specific languages Jackson and Sztipanovits introduce typed predicate logic to handle object types in models [19]. In contrast to Thalheim, they do not adopt the concept of a language structure for model instances, but rather consider a model to be a set of valid statements about the model. This is also true for Telos [24], which builds on the premise that the concepts of entities and links 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 [25]. Similar to Jackson and Sztipanovits knowledge 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 the attempts above a model is constituted by statements, whereas in our approach a model is any language interpretation not explicitly excluded by the statements constraining valid models.

Guizzardi builds a theory of ontologically-driven conceptual modeling based on modal logic and develops in several contributions a comprehensive formal system for conceptualizations of domains as basis for truthful modeling languages, e.g. [16] [17]. In this advanced theory, fruitful for the objective of an ontology-

based, domain-faithful grounding for modeling languages, these languages as well as models and metamodels are a-posteriori concepts implicitly obtained from ontological considerations. In the paper at hand we do not restrict to ontology-driven conceptual modeling and approach models and languages as the a-priori concepts.

Summarizing we conclude, that formal languages in logic prove to be suitable for the intended formalism, as the structure of modeling languages including its linguistic character can be grounded in concepts of formal languages. Therefore, in the work at hand we propose a formal definition of modeling languages in which we concretely point out the modeling concepts and their formal equivalent in logical terms with prospect of successive elaboration. This paper extends our prior work in [10].

3 Definition of Formal Modeling Languages

The intended definition shall serve as a cornerstone for a common way of specifying modeling languages, which thereby become 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. Therefore, we adopt the core concepts of modeling languages identified in [23] – object and relation types, attributes, inheritance and constraints – to be the basic constituents of formal modeling languages.

We use typed (also called sorted) predicate logic in this approach. The mathematical basics can be found in textbooks on logic or mathematics for computer science, e.g. [11] [27]. Some remarks on notation: To ease the differentiation between language and model level, we use capital letters for the symbols of the former and lowercase letters for the elements of the latter.

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 $\boldsymbol{C}$ 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 $<_O \subset \mathcal{S}_O \times \mathcal{S}_O$ to indicate the inheritance relation between the corresponding object types;*
- *$\mathcal{F}$ is a set of typed function symbols such that:*
  - *for each relation type $\boldsymbol{R}$ in $\mathcal{S}_R$ there exist two function symbols $F_s^{\boldsymbol{R}}$ and $F_t^{\boldsymbol{R}}$ with domain type $\boldsymbol{R} \in \mathcal{S}_R$ and codomain type $\boldsymbol{O}_s, \boldsymbol{O}_t \in \mathcal{S}_O$ assigning the source and target object types to a relation;*
  - *for each single-value attribute $\boldsymbol{A}$ of an object or relation type $\boldsymbol{T}$ there exists a function symbol $F^{\boldsymbol{A}}$ with domain type $\boldsymbol{T}$ and codomain type an element in $\mathcal{S}_D \cup \mathcal{S}_O \cup \mathcal{S}_R$ assigning the data type or referenced object type or relation type to an attribute;*
- *$\mathcal{R}$ is a set of typed relation symbols;*

- *$\mathcal{C}$ is a set of typed constants to specify the possible values c_i of type $\boldsymbol{T} \in \mathcal{S}_D$ of the attributes;*
- *the set $\boldsymbol{C}$ is a set of sentences in $\mathcal{L}$ constraining the possible models, also called the postulates of the language.*

This definition explicates the specification of the essential modeling concepts of a language, i.e. object types and inheritance, binary directed relation types and single value attributes. Note that the definition does not prohibit the existence of additional symbols in the signature, so broader concepts like n-ary relations or multi-value attributes 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 relations of the same type between the same two objects, as well as for inheritance of relation types.

With the data types and constants we can define attribute domains like integers via specifying a type called $\mathbb{N}$ and constant symbols 0,1,2,3, ... 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 $\mathcal{C}$.

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 $\boldsymbol{C}$, i.e. $\mathcal{M}$ consists of*

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

This definition of models as language structures goes beyond a visualisation and considers models as knowledge structures as described in [5]. 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 inheritance with the definition of relation types in metamodels.

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

Fig. 1. Notation excerpt of the CoChaCo Method [22]

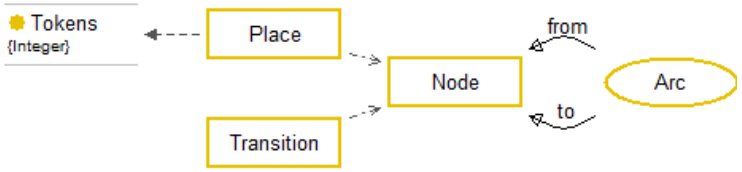

Fig. 2. A metamodel of Petri Nets

[22]. This method comprises concrete syntax for most of the concepts contained in `M2FOL` with a slightly different naming, see Figure 1.

Example 1. The Petri Net Modeling Language $\mathcal{PN}$
The Petri Net metamodel depicted in Figure 2 comprises three object types **Node**, **Place** and **Transition** constituting $\mathcal{S}_O$. Thereby **Place** and **Transition** inherit from **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 $\mathbb{N}$ with the usual addition $+$ and order relation $<_{\mathbb{N}}$ as well as constants in $\mathcal{C} = \{0, 1, 2, ...\}$ all of type $\mathbb{N}$. 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**, $\mathbb{N}$}. For the relation **Arc** we have to specify the source and target object types by introducing two function symbols F_s^{Arc} and F_t^{Arc} both with domain **Arc** and codomain **Node**. For the attribute **Tokens** we introduce a function symbol F^{Tokens} with domain **Place** and codomain $\mathbb{N}$ assigning each place instance a number of tokens. Finally we have to define the constraints of the language. These rules are not contained in a graphical metamodel, but usually specified in an additional source like OCL or simply stated in natural language. In the predicative formalization they 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** (1), as well as the alternation of types of the elements connected by an arc (2, 3) and the prohibition of multiple arcs between the same two elements (4). For ease of readability we abuse the notation $\forall x \in \mathbf{T}$ for x being of type **T** instead of using the type specific quantifier $\forall_{\mathbf{T}} x$.

$$\forall x \in \mathbf{Node}\ \exists y \in \mathbf{Place}, z \in \mathbf{Transition}\ (x = y \vee x = z) \tag{1}$$

$$\nexists x, y \in \mathbf{Place}, u \in \mathbf{Arc}\ (F_s^{Arc}(u) = x \wedge F_t^{Arc}(u) = y) \tag{2}$$

$$\nexists x, y \in \mathbf{Transition}, u \in \mathbf{Arc}\ (F_s^{Arc}(u) = x \wedge F_t^{Arc}(u) = y) \tag{3}$$

$$\forall u, v \in \mathbf{Arc}\ (F_s^{Arc}(u) = F_s^{Arc}(v) \wedge F_t^{Arc}(u) = F_t^{Arc}(v) \implies u = v) \tag{4}$$

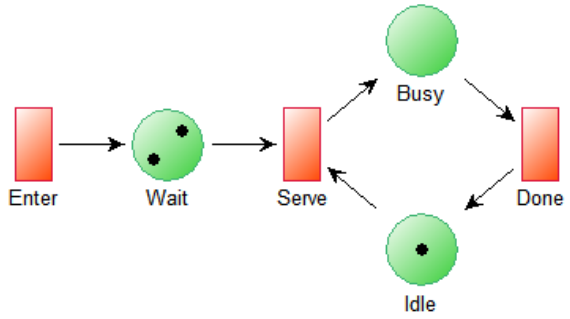

Fig. 3. A Petri Net model depicting a simple barber shop scenario

Example 2. A Petri Net Model
A Petri Net model depicting a simple barber shop scenario is shown in Figure 3. Its formalization, i.e. the corresponding $\mathcal{PN}$-structure, looks as follows: The universe of places $\mathcal{U}_P$ contains three elements $\mathcal{U}_P = \{\mathbf{w}(ait), \mathbf{b}(usy), \mathbf{i}(dle)\}$. The universe of transitions $\mathcal{U}_T$ comprises three elements $\mathcal{U}_T = \{\mathbf{e}(nter), \mathbf{s}(erve), \mathbf{d}(one)\}$. Six arc elements exist in $\mathcal{U}_A = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_5, \mathbf{a}_6\}$ with source and target $f_s^{Arc}(\mathbf{a}_1) = \mathbf{e}$, $f_t^{Arc}(\mathbf{a}_1) = \mathbf{w}$, $f_s^{Arc}(\mathbf{a}_2) = \mathbf{w}$, $f_t^{Arc}(\mathbf{a}_2) = \mathbf{s}$, $f_s^{Arc}(\mathbf{a}_3) = \mathbf{s}$, $f_t^{Arc}(\mathbf{a}_3) = \mathbf{b}$, $f_s^{Arc}(\mathbf{a}_4) = \mathbf{b}$, $f_t^{Arc}(\mathbf{a}_4) = \mathbf{d}$, $f_s^{Arc}(\mathbf{a}_5) = \mathbf{d}$, $f_t^{Arc}(\mathbf{a}_5) = \mathbf{i}$, $f_s^{Arc}(\mathbf{a}_6) = \mathbf{i}$, and $f_t^{Arc}(\mathbf{a}_6) = \mathbf{s}$. For the attribute type and values the natural numbers $\mathbb{N}$ are included in the model, $\mathcal{U}_{\mathbb{N}} = \{0, 1, 2, ...\}$. The instantiation of the attribute *Tokens* looks as follows: $f^{Tokens}(\mathbf{w}) = 2$, $f^{Tokens}(\mathbf{b}) = 0$ and $f^{Tokens}(\mathbf{i}) = 1$. We can easily check that the formalized model satisfies all postulates 1–4 of the language $\mathcal{PN}$.

4 M2FOL – MetaModel 2 First Order Logic A Formal Modeling Language for Metamodels

According to the principle of metaization [38] metamodels are models themselves conforming to a metamodeling language which is itself a modeling language, see Figure 4. We propose a formal modeling language in the sense of Definition 1 for metamodels. This language is capable of describing exactly the concepts explicated in Definition 1.

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 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 *inheritance* 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 *inheritance*, *from*, and *to*, furthermore require assignment functions for source and target specification. Data types and data are necessary for defining attribute domains and its values, e.g. the domain $\mathbb{N}_{0..10}$ and values $\{0, 1, 2, ..., 9, 10\}$ or an enumeration list domain *gender* with values *male*, *female*, *else*. Attribute types need the assignment of owning type and value domain.

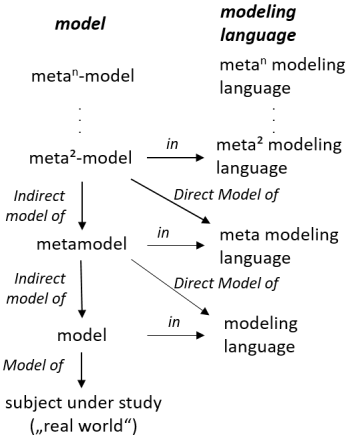

Fig. 4. The metamodeling hierarchy according to Strahringer [38]

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):* $\mathcal{S}_O = \{OT, RT, AT, DT, D\}$*;*
- $\mathcal{S}_R$ *consists of the types* Inh*(eritance),* Fr*(om), and* To*:* $\mathcal{S}_R = \{Inh, Fr, To\}$*;*
- $\mathcal{S}_D$ *is empty;*
- *the set of function symbols consists of following nine elements:*
  - *two symbols* F_s^{Inh} *and* F_t^{Inh} *assigning source and target to* Inh*-typed relations:* $F_s^{Inh} : Inh \to OT$, $F_t^{Inh} : Inh \to OT$*;*
  - *two symbols* F_s^{Fr} *and* F_t^{Fr} *assigning source and target to* Fr*-typed relations:* $F_s^{Fr} : Fr \to RT$, $F_t^{Fr} : Fr \to OT$*;*
  - *two symbols* F_s^{To} *and* F_t^{To} *assigning source and target to* To*-typed relations:* $F_s^{To} : To \to RT$, $F_t^{To} : To \to OT$*;*
  - *two symbols* F_{val} *and* 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 simple value:* $F_{val} : AT \to DT \cup OT \cup RT$, $F_{type} : AT \to OT \cup RT$*;*
  - *a symbol* F_{DT} *to assign a data type to a data element:* $F_{DT} : D \to DT$*;*
- $\mathcal{R}$ *consists of a symbol* $<_{OT}$ *transitively extending the inheritance relation given by* Inh *to a strict partial order on the set of object types:* $\mathcal{R} = \{<_{OT} \subset OT \times OT\}$*.*

The postulates of the language (for brevity we mostly use the abbreviation xry for relation r of type $\boldsymbol{T}$*, x being* $F_s^{\boldsymbol{T}}(r)$ *and y being* $F_t^{\boldsymbol{T}}(r)$*):*

$$\forall x, y, z \in \mathit{OT}, v, w \in \mathit{Inh}\ (xvy, xwz \implies y = z \wedge v = w) \tag{5}$$

$$\forall x, y \in \mathit{OT}, u \in \mathit{Inh}\ (xuy \implies x <_{\mathit{OT}} y) \tag{6}$$

$$\forall x, y \in \mathit{OT}\ \exists z \in \mathit{OT}, u \in \mathit{Inh}\ (x <_{\mathit{OT}} y \implies xuy \vee (xuz \wedge z <_{\mathit{OT}} y)) \tag{7}$$

$$\forall x \in \mathit{RT}\ \exists y, z \in \mathit{OT}, u \in \mathit{Fr}, v \in \mathit{To}\ (xuy \wedge xvz) \tag{8}$$

$$\nexists u, v \in \mathit{Fr}\ (F_s^{\mathit{Fr}}(u) = F_s^{\mathit{Fr}}(v) \wedge u \neq v) \tag{9}$$

$$\nexists u, v \in \mathit{To}\ (F_s^{\mathit{To}}(u) = F_s^{\mathit{To}}(v) \wedge u \neq v) \tag{10}$$

For $<_{\mathit{OT}}$ *we furthermore require to be a strict partial order, i.e.* $<_{\mathit{OT}}$ *is transitive, irreflexive and antisymmetric.*

The constraints ensure single inheritance (5), $<_{\mathtt{OT}}$ being the transitive closure of `Inh` under the assumption that $\mathcal{U}_{\mathtt{OT}}$ is finite (6–7), and the existence and uniqueness of to and from objects of a relation (8-10). The absence of cyclic inheritance and self-inheritance follow from the properties of $<_{\mathtt{OT}}$.

With this language we now can transfer the graphical metamodel of Figure 2 to a formal M2FOL-model.

Example 3. The Petri Net Metamodel $\mathtt{M}_{NP}$
The universe of object types $\mathcal{U}_{\mathtt{OT}}$ comprises three elements `n`(ode), `p`(lace), and `tr`(ansition). The universe of relation types $\mathcal{U}_{\mathtt{RT}}$ contains one element `a`(rc). One element `tok`(ens) is contained in the universe of attribute types $\mathcal{U}_{\mathtt{AT}}$. The universe $\mathcal{U}_{\mathtt{Inh}}$ contains the instantiation 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: $\mathtt{f}_s^{\mathtt{Inh}}(\mathtt{p\_n}) = \mathtt{p}$, $\mathtt{f}_t^{\mathtt{Inh}}(\mathtt{p\_n}) = \mathtt{n}$, $\mathtt{f}_s^{\mathtt{Inh}}(\mathtt{tr\_n}) = \mathtt{tr}$, $\mathtt{f}_t^{\mathtt{Inh}}(\mathtt{tr\_n}) = \mathtt{n}$, $\mathtt{f}_s^{\mathtt{Fr}}(\mathtt{a\_from}) = \mathtt{a}$, $\mathtt{f}_t^{\mathtt{Fr}}(\mathtt{a\_from}) = \mathtt{n}$, $\mathtt{f}_s^{\mathtt{To}}(\mathtt{a\_to}) = \mathtt{a}$, $\mathtt{f}_t^{\mathtt{To}}(\mathtt{a\_to}) = \mathtt{n}$. From `Inh` the transitive order relation $<_{\mathtt{OT}}$ is deduced: $<_{\mathtt{OT}} = \{(\mathtt{p}, \mathtt{n}), (\mathtt{tr}, \mathtt{n})\}$. Furthermore there are data values $\{0, 1, 2, ...\}$ in $\mathcal{U}_{\mathtt{D}}$ all of type $\mathbb{N} \in \mathcal{U}_{\mathtt{DT}}$, $\mathtt{f}_{DT}(i) = \mathbb{N}\ \forall i \in \mathcal{U}_{\mathtt{D}}$. These are needed for the value domain of the attribute type `tok`, an attribute assigned to `p`: $\mathtt{f}_{type}(\mathtt{tok}) = \mathtt{p}$, $\mathtt{f}_{val}(\mathtt{tok}) = \mathbb{N}$. In short this can be written as following:

$$\mathcal{U}_{\mathtt{OT}} = \{\mathtt{n}(ode),\ \mathtt{p}(lace),\ \mathtt{tr}(ansition)\},\ \mathcal{U}_{\mathtt{RT}} = \{\mathtt{a}(rc)\},\ \mathcal{U}_{\mathtt{AT}} = \{\mathtt{tok}(en)\}, \tag{11}$$

$$\mathcal{U}_{\mathtt{Inh}} = \{\mathtt{p\_n}, \mathtt{tr\_n}\},\ \mathcal{U}_{\mathtt{Fr}} = \{\mathtt{a\_from}\},\ \mathcal{U}_{\mathtt{To}} = \{\mathtt{a\_to}\} \tag{12}$$

$$\mathcal{U}_{\mathtt{DT}} = \{\mathbb{N}\},\ \mathcal{U}_{\mathtt{D}} = \{0, 1, 2, ...\}, <_{\mathtt{OT}} = \{(\mathtt{p}, \mathtt{n}), (\mathtt{tr}, \mathtt{n})\} \tag{13}$$

$$\mathtt{f}_s^{\mathtt{Inh}}(\mathtt{p\_n}) = \mathtt{p}, \mathtt{f}_t^{\mathtt{Inh}}(\mathtt{p\_n}) = \mathtt{n}, \mathtt{f}_s^{\mathtt{Inh}}(\mathtt{tr\_n}) = \mathtt{tr}, \mathtt{f}_t^{\mathtt{Inh}}(\mathtt{tr\_n}) = \mathtt{n}, \tag{14}$$

$$\mathtt{f}_s^{\mathtt{Fr}}(\mathtt{a\_from}) = \mathtt{a}, \mathtt{f}_t^{\mathtt{Fr}}(\mathtt{a\_from}) = \mathtt{n}, \mathtt{f}_s^{\mathtt{To}}(\mathtt{a\_to}) = \mathtt{a}, \mathtt{f}_t^{\mathtt{To}}(\mathtt{a\_to}) = \mathtt{n}, \tag{15}$$

$$\mathtt{f}_{type}(\mathtt{tok}) = \mathtt{p}, \mathtt{f}_{val}(\mathtt{tok}) = \mathbb{N}, \mathtt{f}_{DT}(i) = \mathbb{N}\ \forall i \in \mathcal{U}_{\mathtt{D}} \tag{16}$$

This formal metamodel $\mathtt{M}_{NP}$ conforms to all constraints 5-10 and describes the formal language $\mathcal{NP}$ introduced in Example 1. Their subordination prompts a

generic procedure on how to deduce the latter one from the former one. In Table 1 we present this procedure as an algorithm.

	M2FOL (Meta)Model to Language-Signature	Mapping
1.	Each metamodel element $\mathtt{o}$ in the set $\mathcal{U}_{\mathtt{OT}}$ defines an object type $\mathbf{O}$ of the language. The inheritance relation $<_{\mathtt{OT}} \subset \mathcal{U}_{\mathtt{OT}} \times \mathcal{U}_{\mathtt{OT}}$ must be adopted to the types.	$\mathtt{o} \in \mathcal{U}_{\mathtt{OT}} \rightarrowtail \mathbf{O} \in \mathcal{S}_O$
2.	Each metamodel element $\mathtt{r}$ in the set $\mathcal{U}_{\mathtt{RT}}$ defines a relation type $\mathbf{R}$ of the language.	$\mathtt{r} \in \mathcal{U}_{\mathtt{RT}} \rightarrowtail \mathbf{R} \in \mathcal{S}_R$
3.	For each relation type $\mathtt{r} \in \mathcal{U}_{\mathtt{RT}}$ there exist an element $\mathtt{s}$ of type From and an element $\mathtt{t}$ of type To and both relation elements have as source element $\mathtt{r}$, $\mathtt{f}_s^{\mathtt{Fr}}(\mathtt{s}) = \mathtt{r}, \mathtt{f}_s^{\mathtt{Fr}}(\mathtt{t}) = \mathtt{r}$. The assignment $\mathtt{f}_t^{\mathtt{To}}(\mathtt{s}) = \mathtt{o}_s$ indicates the source object type of $\mathbf{R}$, $\mathtt{f}_t^{\mathtt{To}}(\mathtt{t}) = \mathtt{o}_t$ indicates the target object type of $\mathbf{R}$.	$\mathtt{r}, \mathtt{s}, \mathtt{o}_s \rightarrowtail \mathtt{F}_s^{\mathtt{R}} : \mathbf{R} \rightarrow \mathbf{O}_s$; $\mathtt{r}, \mathtt{t}, \mathtt{o}_t \rightarrowtail \mathtt{F}_t^{\mathtt{R}} : \mathbf{R} \rightarrow \mathbf{O}_t$
4.	Each metamodel element $\mathtt{dt}$ in $\mathcal{U}_{\mathtt{DT}}$ defines a data type $\mathbf{DT}$ of the language. Each metamodel element $\mathtt{d}$ in $\mathcal{U}_{\mathtt{D}}$ with $\mathtt{f}_{\mathtt{DT}}(\mathtt{d}) = \mathtt{dt}$ becomes a constant symbol C_d in $\mathcal{C}$ of type $\mathbf{DT}$.	$\mathtt{dt} \in \mathcal{U}_{\mathtt{DT}} \rightarrowtail \mathbf{DT} \in \mathcal{S}_D$; $\mathtt{d} \in \mathcal{U}_{\mathtt{D}} \rightarrowtail C_{\mathtt{d}} \in \mathcal{C}$
5.	Each metamodel element $\mathtt{a}$ in the set $\mathcal{U}_{\mathtt{AT}}$ defines a function symbol $\mathtt{F}^{\mathtt{a}}$ of the language. The object or relation type $\mathtt{a}$ belongs to, i.e. the domain of $\mathtt{F}^{\mathtt{a}}$, is given by the assignment $\mathtt{f}_{type}(\mathtt{a}) = \mathtt{t}_{ty} \in \mathcal{U}_{\mathtt{OT}} \cup \mathcal{U}_{\mathtt{RT}}$, its value range, i.e. codomain, by $\mathtt{f}_{val}(\mathtt{a}) = \mathtt{t}_v \in \mathcal{U}_{\mathtt{OT}} \cup \mathcal{U}_{\mathtt{RT}} \cup \mathcal{U}_{\mathtt{DT}}$	$\mathtt{a}, \mathtt{t}_{ty}, \mathtt{t}_v \rightarrowtail F^{\mathtt{a}} : T_{ty} \rightarrow T_v$
6.	The constraints of the language have to be added manually, because this information is not determined by the metamodel.	

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

Finally we take the meta-perspective on M2FOL. Its graphical metamodel is depicted in Figure 5 and contains five objects of type OT={ot, rt, at, dt, d}, three objects of type RT={inh, fr, to}, three objects of type AT= {value_domain, assigned_to, assigned_domain}, three relations of type From={source_inh, source_to, source_fr}, and three relations of type To={target_inh, target_to, target_fr}, furthermore 18 assignments of source and target objects, attribute owning types and attribute value types. On the one hand it is itself a model conforming to the language M2FOL, to be precise the graphical representation thereof. On the other hand this metamodel describes M2FOL and with the algorithm presented above we deduce Definition 3. So we conclude that the proposed modeling language for metamodels M2FOL is self-describing and thereby complete the formalization of the full modeling stack.

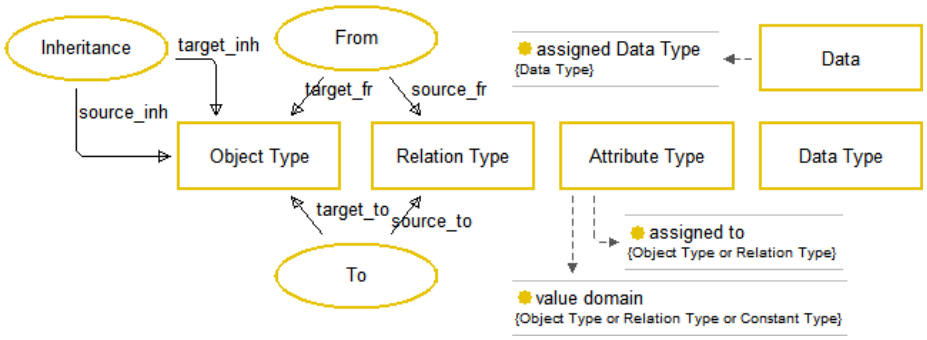

Fig. 5. The metamodel of `M2FOL`

5 Conclusion and Outlook

In this paper we presented 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 the instantiation relation between model and 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 Net 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-descriptive, which can be seen by applying the algorithm to its own metamodel.

This work investigates the theoretical foundations of conceptual modeling. For a practical application of the formalism a suitable tool for transforming graphical metamodels into formal ones can be developed. This formal specification of a language, 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 practice of 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 multi-value attributes, the concepts of mixins and extenders for modular metamodels as proposed in [41], or the structural types of relations identified in [39]. Furthermore, we intend to investigate formal modeling languages $\mathcal{L}$ beyond their syntax and approach also their functionality by describing domain events on models, like firing a transition in a Petri Net, as transitions between $\mathcal{L}$-structures.

Finally, by using a sophisticated mathematical theory as grounding for the definition of modeling languages we can use this knowledge stack as resource

to further establish a formal foundation for modeling languages and approach old problems with new tools. One issue currently under research is the process of language interleaving, which is highly relevant for ensuring consistency of the same information captured in models of different languages or model types. Connecting formal languages is a well researched topic in mathematical logic and provides methods which are now also applicable to modeling languages.

References

1. BOC: ADOxx Metamodelling Platform (2020), https://www.adoxx.org
2. Bork, D., Buchmann, R.A., Karagiannis, D., Lee, M., Miron, E.T.: An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem. Communications of the Association for Information Systems **44**(1), 673–679 (2019)
3. 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 (jan 2014)
4. Bork, D., Karagiannis, D., Pittl, B.: A survey of modeling language specification techniques. Information Systems **87**, 101425 (jan 2020)
5. Buchmann, R.A., Ghiran, A.M., Döller, V., Karagiannis, D.: Conceptual Modeling Education as a "Design Problem". Complex Systems Informatics and Modeling Quarterly (21), 21–33 (dec 2019)
6. 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)
7. 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, Heidelberg (2012)
8. 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 International Publishing, Cham (2018)
9. 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 International Publishing, Cham (2019)
10. 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)
11. Enderton, H.B.: A Mathematical Introduction To Logic. Harcourt/Academic Press, San Diego, 2nd edn. (2001)
12. 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)
13. 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 Heidelberg (2013)

14. 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. Business and Information Systems Engineering **6**, 39–43 (2014)
15. Guarino, N., Guizzardi, G., Mylopoulos, J.: On the Philosophical Foundations of Conceptual Models. In: Proceedings of the 29th International Conference on Information Modelling and Knowledge Bases, EJC 2019. Frontiers in Artificial Intelligence and Applications, vol. 321, pp. 1–15. IOS Press (2019)
16. 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)
17. Guizzardi, G.: Logical, ontological and cognitive aspects of object types and cross-world identity with applications to the theory of conceptual spaces. In: Applications Of Conceptual Spaces: The Case For Geometric Knowledge Representation, pp. 165–186. Springer International Publishing (jan 2015)
18. 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)
19. Jackson, E., Sztipanovits, J.: Towards a Formal Foundation for Domain Specific Modeling Languages. In: Proceedings of the 6th ACM & IEEE International Conference on Embedded Software. pp. 53–62. EMSOFT '06, ACM, New York, NY, USA (2006)
20. 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, Heidelberg (2006)
21. 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 Verlag (2018)
22. 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)
23. 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)
24. 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. Requirements Engineering (2020)
25. Koubarakis, M., Mylopoulos, J., Stanley, M., Borgida, A.: Telos: features and formalization. Tech. rep., Technical report KRRTR- 89-4, Department of Computer Science, University of Toronto (1989)
26. 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 (nov 2016)
27. Mazzola, G., Milmeister, G., Weissmann, J.: Comprehensive mathematics for computer scientists 1: Sets and numbers, graphs and algebra, logic and machines, linear geometry (second edition). Springer Berlin Heidelberg (2006)
28. Mouratidis, H., Giorgini, P.: Secure Tropos: A security-oriented extension of the Tropos methodology. In: International Journal of Software Engineering and Knowl-

edge Engineering. vol. 17, pp. 285–309. World Scientific Publishing Company (apr 2007)
29. Olivé, A.: Conceptual modeling of information systems. Springer-Verlag, Berlin Heidelberg (2007)
30. 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 (aug 2019)
31. 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, Heidelberg (nov 2013)
32. 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 International Publishing, Cham (2017)
33. Rrenja, A., Matulevičius, R.: Pattern-based security requirements derivation from secure tropos models. In: Ralyté, J., España, S., Pastor, O. (eds.) The Practice of Enterprise Modeling. PoEM 2015. vol. 235, pp. 59–74. Springer Verlag (2015)
34. 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 Verlag (2016)
35. Sandkuhl, K., Fill, H.G., Hoppenbrouwers, S., Krogstie, J., Matthes, F., Opdahl, A., Schwabe, G., Uludag, Ö., Winter, R.: From Expert Discipline to Common Practice: A Vision and Research Agenda for Extending the Reach of Enterprise Modeling. Business and Information Systems Engineering **60**, 69–80 (2018)
36. 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 (nov 2019)
37. 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 International Publishing, Cham (2017)
38. Strahringer, S.: Zum Begriff des Metamodells. Schriften zur Qualitativen Betriebswirtschaftslehre, Techn. Hochsch., Inst. für Betriebswirtschaftslehre (1995)
39. Thalheim, B.: Towards a theory of conceptual modelling. In: Conceptual Modeling. ER 2009. Lecture Notes in Computer Science. vol. 5833 LNCS, pp. 45–54. Springer, Berlin, Heidelberg (2009)
40. 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 Heidelberg (2011)
41. Zivkovic, S., Karagiannis, D.: Mixins and Extenders for Modular Metamodel Customisation. In: Proceedings of the 18th International Conference on Enterprise Information Systems. vol. 1, pp. 259–270. SCITEPRESS - Science and and Technology Publications (2016)