Software and Systems Modeling
https://doi.org/10.1007/s10270-022-01047-4

SPECIAL SECTION PAPER

MetaMorph: formalization of domain-specific conceptual modeling methods—an evaluative case study, juxtaposition and empirical assessment

Victoria Döller[1] · Dimitris Karagiannis[1] · Wilfrid Utz[2]

Received: 21 November 2021 / Revised: 6 September 2022 / Accepted: 12 September 2022

Abstract
Models have evolved from mere pictures supporting human understanding and communication to sophisticated knowledge structures processable by machines and establish value through their processing capabilities. This entails an inevitable need for computer-understandable modeling languages and causes formalization to be a crucial part in the lifecycle of engineering a modeling method. An appropriate formalism must be a means for providing a structural definition to enable a theoretical investigation of conceptual modeling languages and a unique, unambiguous way of specifying the syntax and semantics of an arbitrary modeling language. For this purpose, it must be generic and open to capturing any domain and any functionality. This paper provides a pervasive description of the formalism MetaMorph based on logic and model theory—an approach fulfilling the requirements above for modeling method engineering. The evaluation of the formalism is presented following three streams of work: First, two evaluative case studies illustrate the applicability of MetaMorph formalism concept by concept on the modeling language ProVis from the domain of stochastic education and the well-known Entity-Relationship language. ProVis as well as ER comprise only a few objects and relation types but with high interconnection and expressive power and are therefore considered interesting specimens for formalization. Second, a comprehensive juxtaposition of MetaMorph to three other formalization approaches based on different foundational theories is outlined concept by concept to underpin the formalism design. Third, an empirical evaluation has been performed, assessing the usability and adequacy of the formalism within a classroom assessment. The results allow for conclusions on the completeness, intuitiveness, and complexity as well as on interdependencies with engineers' skills.

Keywords Conceptual modeling · Agile modeling method engineering · Domain-specific modeling language · Formal language · Logic · Evaluation

1 Formalization in conceptual modeling

Conceptual modeling methods are an established approach for representing complex information in a clear and unambiguous way. Once meant as a means for understanding and communication between humans [44], the field of application of modeling methods was considerably broadened. Nowadays, models are considered as processable, exploitable knowledge structures [6] and are supported by diverse functionality such as simulation, transformation, or reasoning, going beyond a mere visual representation of information [3]. This amplifies the value of models and implies an imminent need for technical support for model processing and hence, results in the requirement for the formalization of modeling languages as processing functionality has to be considered on the language level.

Communicated by Iris Reinhartz-Berger, Jelena Zdravkovic, and Asif Gill.

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

Dimitris Karagiannis
dimitris.karagiannis@univie.ac.at

Wilfrid Utz
wilfrid.utz@omilab.org

[1] Research Group Knowledge Engineering, Faculty of Computer Science, University of Vienna, Währinger Str. 29, 1090 Vienna, Austria

[2] OMiLAB NPO, Lützowufer 1, 10785 Berlin, Germany

Published online: 06 October 2022

For a couple of years now, the research community in conceptual modeling started an endeavor to collate and structure the extensive knowledge stack on conceptual modeling from practical and theoretical investigations and compose them into a sound and generally accepted basis of conceptual modeling research, i.e., a precision of the notion of involved concepts [13,23,28,38,41], a systematization the research field [13,14,52], building a comprehensive theory of conceptual modeling [53,57], as well as a research agenda for CM [49,56]. The goal is to strengthen the sovereign research field of conceptual modeling detached from a concrete domain, e.g., information systems engineering or enterprise modeling, and enhance the maturity of the field of conceptual modeling as a science. These considerations regularly postulate the need for an investigation of the formal foundations of conceptual modeling [4,14,27,56]. A mature comprehension of the research field includes a concise analysis of the formal, structural nature of conceptual modeling languages and targets an integrative formal foundation providing answers to the question: *What exactly is a conceptual modeling language from a formal, structural point of view?* This allows for the study of modeling languages and their properties and characteristics on a level of formality similar to database theory and database normalization.

A formalism in the above sense must comprise the relevant concepts of languages and provide an integrative foundation open for affiliation of any progression in conceptual modeling, whether advanced concepts or sophisticated functionality. It must be generic and admit the formalization of any modeling language designed using the prevalent building blocks of conceptual modeling. A domain-specific or even language-specific formalism might meet the needs of a precise, computer-processable specification but does not provide a formal foundation for the investigation of the structural nature of modeling languages in general. It also multiplies the amount of work if the development of a formalism has to be done for each domain or language individually.

Also, for practicing conceptual modeling, a maturation of the scientific field and a suitable formal foundation are beneficial. With a more thorough knowledge base, a commonly accepted, well-founded formal notion, and a common use and practice of modeling, the quality of modeling methods increases, and the design and development of languages professionalizes and becomes more sustainable. An elaborate formal foundation unifying the groundwork of conceptual modeling methods makes it easier to connect models and languages and build the big picture of a system under study as an ensemble of a whole suite of models. It serves practitioners as a toolbox providing established approaches and methods, e.g., for language interleaving and consistency or model transformation [16].

This positive effect on the practice of conceptual modeling, of course, can only be achieved if the application of the formalism is not a barrier but an encouragement to method engineers. The formalism must be intuitive to use and reasonable in terms of effort. The practitioner's experience justifies or dissents the quality and adequate compliance of the formalism to the character of conceptual modeling languages as utilized by engineers working in diverse domains contributing various perspectives and expectations on conceptual modeling.

The considerations above led us to the proposition of four concrete requirements for an appropriate formalism [16]: (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, and (4) it must provide an integrative formal foundation offering canonical tools for the advancements in conceptual modeling research.

In an exhaustive literature review, we could not find a suitable approach meeting all these requirements, see Sect. 3. Therefore, we intended to close this gap and developed the MetaMorph formalism comprehensively presented in [16]. The objective of the paper at hand is to complement the theoretical considerations of this prior work on the potential of MetaMorph with a thorough evaluation both from an analytical perspective—by conducting case studies and contrasting it to existing approaches—and from an empirical perspective—by gathering feedback from practitioners. This evaluation is guided by the research questions of the appropriateness of the chosen structural foundation and the appropriateness for the needs and intuition of practitioners. We demonstrate the applicability of MetaMorph to full-fledged modeling languages rather than presenting specific, downscaled examples. Furthermore, we provide a detailed juxtaposition to three other approaches with diverse foundational theories and outline the benefits and drawbacks of these foundations. Finally, we discuss an empirical evaluation of MetaMorph conducted within a classroom assessment to analyze the practical applicability, adequacy, and intuitiveness of the formalism. We intend to fortify by this evaluation the appropriacy of our formalism. Furthermore, the paper serves as guidance for others formalizing their own modeling method. The paper at hand extends our contribution [18] presented at the EMMSAD 2021 conference.

The remainder of this paper is structured as follows: In Sect. 2 we give a more detailed discussion of notion and background of formalization in conceptual modeling. Section 3 summarizes the results of related work to MetaMorph and introduces and discusses existing formalization approaches, establishing the foundation for the work performed. In Sect. 4 the definitions of formal modeling languages and models are introduced, and we provide a discussion of the meta^{2}concepts included in MetaMorph. In Sect. 5, the formalism is evaluated by applying it to the modeling methods ProVis from the area of stochastic education and the Entity-Relationship

modeling language, illustrating all aspects of the definition. Sect. 6 presents a detailed juxtaposition of METAMORPH to three existing approaches, i.e., the FDMM formalism [21], graph grammars [26], and the FORMULA formalism [30]. In Sect. 7 we provide insights into the empirical assessment performed, summarized in Sect. 8, that identifies lessons learned from case studies and the empirical evaluation. Finally, in Sect. 9 we give an outlook on the limitations, future development, and research objectives concerning METAMORPH.

2 Problem discussion and background

2.1 Domain-specific conceptual modeling and agility

Besides the established standardized and general-purpose languages, such as the Unified Modeling Language (UML) and the Business Process Model and Notation (BPMN), modeling has proven beneficial in a variety of domains such as mechanical engineering or enterprise modeling. Formalization is especially important in the light of the emergent practice of agile development of these domain-specific modeling languages (DSML) [22,32] that provide the distinct vocabulary for a domain. Domain-specific languages rely on a fast evolution as novel developments on conceptual and technical level need to be reflected. This permanent evolution once was the reason for the development of metamodeling concepts and platforms to allow for an adaption of the by then static metamodels [35] and is underpinned by the ongoing creation of new domain-specific modeling languages. In the OMiLAB community [47] alone, 45 new methods have been created in the last 6 years and presented in two books: [36,37]. The requirement of the adaption of existing modeling methods to the needs of an organization or individual project even triggered a whole research subfield of conceptual modeling called situational method engineering [29]. This evolution calls for the support of linking and extending existing languages and for identifying suitable processing algorithms and exposes the need to establish a generic formalism of conceptual modeling languages. An established common practice on how to specify, formalize and develop modeling languages unambiguously will support the comparability, reusability, and modularizability of the results.

Furthermore, the design process of a modeling method benefits from a maturation of the formal basics, as a formalization of languages leads to a clear-up of inaccuracies and might reveal misconceptions. Formalization may cause an adaption of language design because it enforces a conclusive walk-through of its conceptionalization. This, in the further course, enables the identification of frequently occurring weaknesses in the design and the establishment of design guidelines and paradigms for the development process.

A process model for DSML development is proposed in the Agile Modeling Method Engineering (AMME) lifecycle [32], see Fig. 1. The need for formal, computer-processable modeling languages is reflected in the AMME lifecycle by stating formalization as an integral phase in the development of a modeling method. The formalization phase targets the refinement and precise specification of a modeling language using an appropriate formalism to support implementations across various metamodeling platforms. A common way of formalizing modeling methods allows for a platform-independent yet computer-processable specification of these languages. This implies that the formalized representation of the modeling method could enable algorithmic solutions to support the design, development, and deployment of the modeling method on the tool level, e.g., using platform-specific translators to transform the platform-independent specification to a platform-specific implementation. A formalism bridging the gap from method design to method implementation in the AMME lifecycle must be as generic as the framework itself: the formalism must not be restrictive in its domain of application or the supported functionality and serves as a tool in the process of agile method development.

2.2 Notion of formalism

According to the need for a formal foundation grounded on a suitable structural theory, the notion of formalism used throughout this paper goes beyond the usually named unambiguity of specifications [4]. We expect a formalism to be based on a suitable formal, structural theory canonically reflecting the constructs of conceptual modeling. Such a foundation offers means to investigate the features of modeling languages defined as structures according to the theory, thereby offering a whole knowledge stack and methods to approach them. This strong requirement distinguishes our approach from other attempts considering a formalism as means to provide a formal syntax for the specification of languages, e.g., the Meta-Object Facility (MOF) standard.

More specifically, the difference is as follows: A formalism is able to capture the underlying structure of a conceptual modeling method, whereas a formal syntax system offers only a unique way of specification. This can be compared to the concept of a graph and the diverse methods for recording a graph, e.g., in a graphical manner, as an adjacent matrix, etc. The underlying concept of a graph is the actual notion carrying the semantics of the structure. The different ways of specification do not provide these semantics in themselves and give no means to investigate the underlying formal structure. This also implies that a formalism in our sense always contains a (at least one) formal syntax stemming from the underlying theory, as the structure has to be recorded in a way.

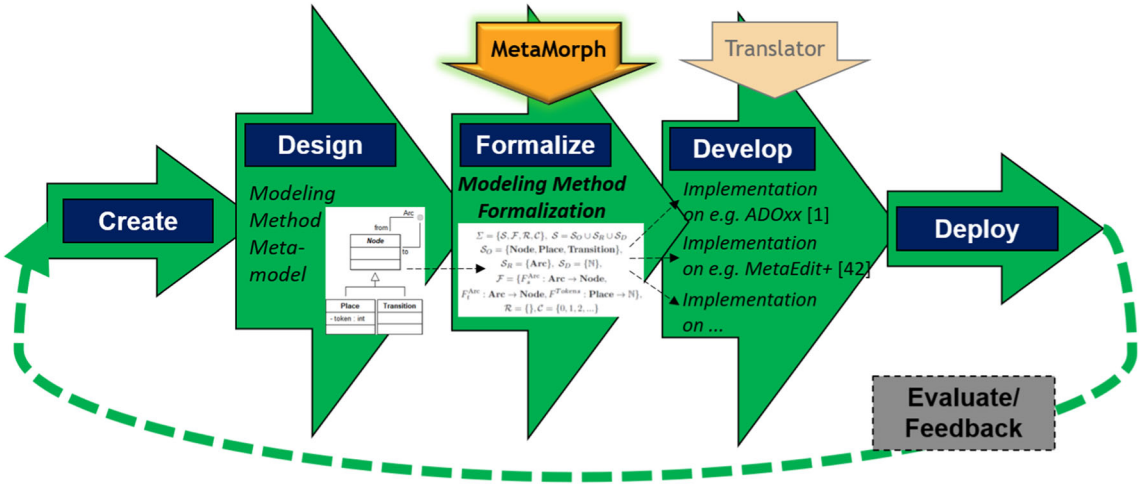


Fig. 1 The AMME lifecycle of modeling methods [32]

2.3 Contextualization of our approach

Figure 2 contextualizes our approach to provide a formal definition space for modeling methods. Besides the layer of formalization, we position the layers of a formal specification as described above and the layer of formal implementation in a concrete platform-specific representation. This positioning is introduced in line with the triptych allegory proposed by Mayr and Thalheim [41]. Within their work, they define conceptual modeling as a tripartite consisting of an *encyclopedic dimension* to base definitional semantics in an ontology or concept space, a *language dimension* to clarify the definition of the language vocabulary, and the *conceptual modeling dimension* in between as a mediation layer. Our formalism is located in the language dimension, taking the language engineering perspective.

3 Related work

We use the triptych of conceptual modeling [41] to delineate our formalism from existing approaches with different purposes. We acknowledge that in the encyclopedic dimension, there exist various elaborate attempts to formalization, like the KL-ONE family [5], Description Logic [2], or approaches using conceptual graphs to formalize

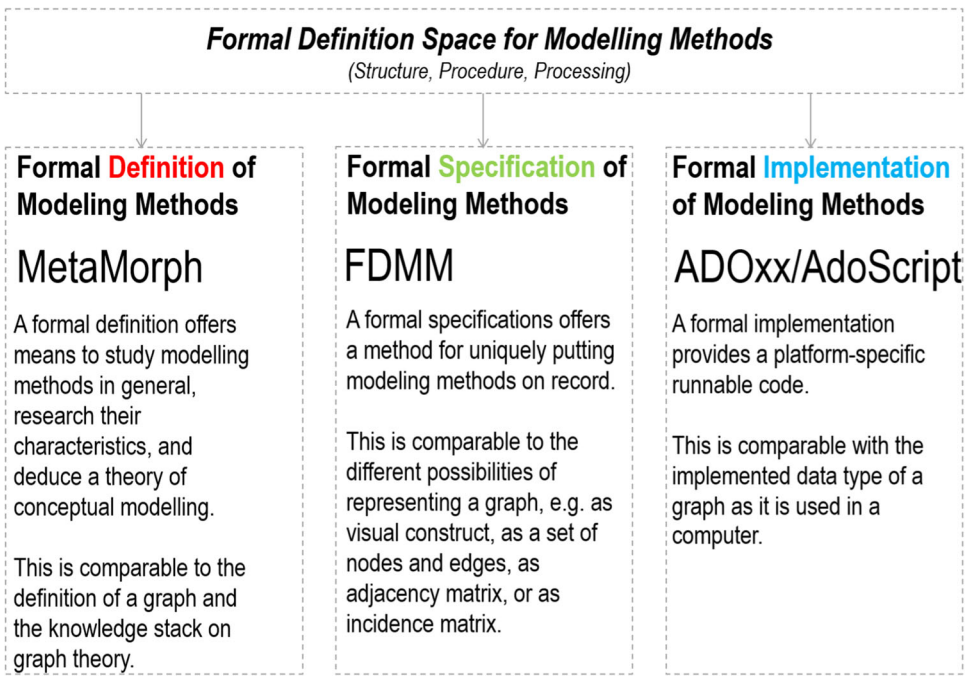


Fig. 2 Layers of Formalization: Definition in [16], Specification in [21] and Implementation in [1]

design concepts [33,58]. Also, the approaches postulating ontology-driven conceptual modeling, e.g., [25], themselves profoundly formalized, have to be ascribed to this dimension. They, on principle, start with a domain conceptualization (and its formalization) and consider modeling languages as a posteriori artifacts derived from a domain ontology (preferably via an isomorphism). Indeed modeling languages are assumed to be formal languages with vocabulary, but it is out of scope to give a closed, concise definition of a formal modeling language with all its components and to investigate its characteristics.

According to the conceptual modeling research framework [14] the approach at hand has to be located in the dimension *Formalize* working on the level of *Conceptual Modeling Languages* and *Metamodeling Languages*. In contrast to that, we have to delineate our formalism from the various attempts addressing the formalization of a specific modeling language. These attempts mostly aim at supporting a concrete domain, purpose, or functionality and do not provide means to define arbitrary languages, and to offer a structural investigation. Examples of formalisms for concrete modeling languages are the OSM-logic for modeling time-dependent system behavior [12], or the SAVE method for simulating IoT systems [11]. Other approaches targeting the formalization of particular aspects of conceptual modeling, such as a formalization of multi-level modeling in [8], explicitly exclude language engineering perspective from their focus and practice conceptual modeling formalization completely independent of linguistic considerations central to the paper at hand.

In the generic considerations on conceptual modeling from [27] and [46] a variety of mathematical concepts are used to describe the structure of different aspects and components of modeling languages. Henderson-Sellers [27] introduces morphisms, functions, relations, and power sets. Olivé [46] concludes each chapter with a summary of contents recorded in logical notion. The intention of both works is to give a precise and formal outline of notions in conceptual modeling but not to give a closed definition of modeling languages and models. They do not aim at a comprehensive formalism and a common procedure of formalization.

When assessing the formalizations in the language dimension of the triptych, existing approaches can be categorized according to the underlying theory they apply, typically graph theory (e.g., KM3 by Jouault and Bezivin [31], graph grammars [26], or a formalization of MOF based on graph morphisms by Weisemöller and Schürr [59]), set theory (e.g., FDMM introduced by Fill et al. [21], or a set-theoretic formalization of Ecore/EMOF proposed by Burger [7, 2.3.2]) or logic (examples below). All these theories provide concepts for the concrete structural behaviour of language constructs. In the evaluation of the applicability of these theories, we can recognize that neither graph theory nor set theory does justice to the linguistic character of modeling languages. They do not provide canonical counterparts for the definition of the instantiation relation between language (or alphabet) and language instance, i.e., the model, an essential characteristic of modeling languages. Indeed, not all of the named approaches aim at a formalism as we defined it. The goal of FDMM, for example, is simply to provide a formal syntax. To underpin the divergent suitability for modeling language formalization, we present a detailed juxtaposition of four exemplifying approaches of the different theories in Sect. 6.

Surveying scientific literature for a suitable structural theory for conceptual modeling reveals increasing attention for logic: modeling languages comprise all characteristics of formal languages [13,23,46,48,55]. Formal languages, as defined in mathematical logic, canonically reflect the prominent characteristics of modeling languages, i.e., the linguistic character providing a vocabulary and the instantiation relation between the meta-layer and the model-layer, and they provide a rich knowledge base about their properties. Based on these results, we consider modeling languages as a subclass of formal languages.

In their investigation of formal foundations of domain-specific languages, Jackson and Sztipanovits [30] introduce term algebras to handle models. They adopt the notion of formal languages with a signature and an alphabet for modeling languages. 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. This hampers the applicability to classical language engineering projects as there is no direct procedure to define object and relation types, attributes, etc. Therefore, the approach lacks the levels of metamodel and model central to conceptual modeling.

Semeráth et al. [50] propose a method to transform modeling languages to first-order logic for model validation with SAT and SMT solvers. Unfortunately, the approach does not contain a comprehensive definition of formal modeling languages in first-order logic (FOL), but it provides a description of transforming EMF metamodels to FOL, giving a glimpse of a possible formalization procedure.

Telos [40,45] builds on the premise that the concepts of entities and relations are omitted and replaced by propositions constituting the knowledge base. Knowledge in Telos is represented as a set of sentences in the formal language. Again, the approach diverges from the core constituents of conceptual modeling. 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.

In his work on the theory of conceptual models, Thalheim [55] describes modeling languages as being based on a signature Σ comprising a set of sentences expressed with

elements of Σ constraining the language. Models are defined as language structures satisfying the sentences. 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 independent of a concrete language.

To accomplish the requirements of conceptual modeling languages, the formalization approach MetaMorph was developed and presented in [15], and refined in [16]. This formalism comprises the core meta^{2}concepts as outlined in [39] and [46]. With MetaMorph we are able to describe conceptual modeling languages as a subclass of formal languages and approach them with established tools and methods from mathematical logic and model theory.

4 Conceptionalization of the MetaMorph formalism

To establish a definition of formal modeling languages, we first have to examine what concepts have to be included in this definition. We, therefore, use a survey conducted by Kern et al. [39] investigating six established metamodeling platforms for the incorporated concepts in the corresponding meta^{2}models. The approach at hand consolidates all concepts detected in at least half of the platforms: object types, relation types (binary), attributes (multi-value), inheritance or specialization (for object types), and a constraint language.

An *object type* is an abstraction of a set of elements in a domain with equal features, constraints, and relationships. Classification defines an instantiation relation between object type and object. *Specialization* (sometimes called inheritance) is a relation between a super- and a subtype where the subtype inherits the defining properties of the supertype and owns some additional ones. *Relation types* are similar to object types abstractions of a set of connections between elements in a domain carrying the same features and constraints with the specific feature of being existentially dependent on the linked objects. They have an arity, i.e., a number of elements connected by the relation. *Attributes* are means to capture single features of an object- or relation type and hold a concrete value on model level. The value belongs to the value domain of the attribute, i.e., the admissible set of possible manifestations an attribute might have. *Constraints* allow defining conditions for the concepts mentioned above, which have to be fulfilled in a model.

These language components mainly coincide with the building blocks mentioned in Olivé's book on conceptual modeling of information systems [46] and are therefore endorsed both from the viewpoint of technical realization as well as from the theoretical viewpoint. Concepts for future integration are specialization of relations, n-ary relations, model types [39], and derived types [46]. The principal integrability of MetaMorph was demonstrated by incorporating the concept of power types outlined in [16].

In the following, we replicate the definition for a formal modeling language as well as for a model formulated in a concrete modeling language. This definition is based on [15] and was refined in [16]. We use typed (also called sorted) first-order predicate logic and build on the mathematical background of model theory. The basics of FOL and model theory can be found in textbooks on logic or mathematics for computer science, e.g., [9,20].

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 i-th 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 types $\mathbf{R} \in \mathcal{S}_R$ 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 constraints or postulates of the language.

Notice that relation types are defined as independent modeling elements with two function symbols to specify their

source and target object instead of defining them as tuples of object elements. This is motivated by the relevance assigned to relations as information carriers in models. Furthermore, this allows for attributes on relations as well as multiple relations between the same two object elements.

When defining attributes, existing language specifications often refer to basic data types and enumerations, such as in programming. This attempt is generalized in the definition at hand by introducing attribute-value types. They are meant to name the whole range of possible values of an attribute. The concrete values possibly adopted by an attribute are defined as constants assigned to the corresponding attribute value type. An example for this is the value type $\mathbb{N}$ with constants $1, 2, 3, \ldots$ or the value type *gender* and constants *male, female, other*.

Constraints are an integral part of METAMORPH. This means we do not need any additional syntax such as, e.g., OCL to formalize the restrictions but have the expressive power of predicate logic at our disposal.

When interpreting modeling languages as formal languages, we get a canonical definition for models as structures of a concrete language.

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 $\mathbf{T}_1 \times \cdots \times \mathbf{T}_n$ and codomain type $\mathbf{T}$ a function $f : \mathcal{U}_{\mathbf{T}_1} \times \cdots \times \mathcal{U}_{\mathbf{T}_n} \to \mathcal{U}_\mathbf{T}$;
- An interpretation of the relation symbols in $\mathcal{L}$, i.e., for each relation symbol $R \in \mathcal{R}$ with domain type $\mathbf{T}_1 \times \cdots \times \mathbf{T}_m$ a relation $r \subseteq \mathcal{U}_{\mathbf{T}_1} \times \cdots \times \mathcal{U}_{\mathbf{T}_m}$;
- 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., $\mathcal{M} \models \phi$.

Note that the specification of a model using the definition above is a supplementary way of representation to the usual graphical way of recording a model. An advantage of this representation is the completeness of information. This means that also attribute values are concretely specified, a detail that is often not visible in a graphical model.

The design of a modeling language is part of the language engineer's craftsmanship and is not prescribed by the formalism. The semantic analysis and construction of language concepts are up to the discretion of the language engineer, who has to determine the intrinsic notion of the domain. An ontological founded perspective might assist in this task, e.g., using the ontology-driven conceptual modeling approach based on the Unified Foundational Ontology (UFO) [25]. UFO was initially presented in [24] where the correspondence between UFO concepts and the core concepts in conceptual modeling also constituting METAMORPH are outlined (object types $\widehat{=}$ universals, relation types $\widehat{=}$ relation universals or relator universals, attributes $\widehat{=}$ properties, and attribute domains $\widehat{=}$ quality dimensions [24, 6.3]).

To illustrate the definitions of formal modeling languages and models and show the feasibility of METAMORPH, we illustrate its application on two extensive case studies in the following sections.

5 Two case studies—proof of concept

In the following, we present two case studies that demonstrate the procedure of formalizing a modeling language concept by concept. The languages are formalized to their full extent without reducing them to small, illustrating examples. This proves the capability of METAMORPH for complete formalization, not restricting to selective parts of a language.

We chose for the case studies the Entity-Relationship modeling language [10] and the modeling language PROVIS from stochastic education [17]. Both of them are expressive, powerful tools comprising only a couple of semantically rich concepts and a row of sophisticated constraints. This makes them perfect examples for formalization. Of course, also languages with an extensive number of concepts can be formalized with METAMORPH, e.g., BPMN or UML. Full formalizations of such languages are mostly created for the purpose of computer-processing but not for human consumption, as the complete signatures become vast. Nevertheless, a formalized modeling language record usually exceeds a natural language specification document in conciseness and precision. Also, the formalization of huge models is not primarily meant for human utilization as it lacks the primary benefit of visual models, the graphical (two-dimensional) depiction of a system under study. The graphical and the formal representation of a model are two different ways of representing the same circumstance, both with different merits, easy comprehensibility for the first one, and completeness and precision for the second one. These considerations make

exhaustive languages desirable to be formalized to become computer-processable but rather unsuitable for case studies aiming at a deeper understanding and demonstration of the presented formalism for language engineers.

Further case studies demonstrating concrete assets of our formalism are an excerpt of UML class and sequence diagrams, the Petri Net Modeling language, as well as a simple process modeling language in [15,16].

5.1 Formalizing the ProVis modeling language

In this case study, we use as demonstration language ProVis—Probability Visualized, a language for digitizing the visualization methods tree diagrams and unit squares for stochastic education. The language was designed and implemented for the use in Austrian schools at the secondary level. For a detailed description of the language and the mathematical background see [17].

The metamodel of ProVis is shown in Fig. 3 using the notation of CoChaCo [34]. This language contains only a few object and relation types, but each of them carries a lot of information in interdependencies and functionality. Therefore, ProVis provides an interesting specimen for formalization.

In Fig. 4 we see examples of a unit square (a) and a tree diagram (b). A unit square is an object on its own, whereas a tree diagram is a construct constituted from several events (rectangles) and transitions (arrows) between them.

Unit squares deal with the occurrence of characteristics in a cohort, each of them with two possible values. An example of characteristics are the smoking behavior and nationality with values *smoker/non-smoker* and *Austria/Germany* in the population of Austria and Germany. Another example is the gender and graduation with values *male/female* and *master's degree/doctoral degree* in the cohort of graduates at University of Vienna in 2019 from Fig. 4a. The frequencies of these values are depicted in the unit square, both as numbers and encoded in the size of the rectangles, each rectangle corresponding to one possible combination of value combinations, e.g., Austrian smokers or male doctoral graduates. Besides the size also the height and width of the four rectangles carry information about the relative proportion of subgroups to each other and to the whole cohort. The difference in heights of the upper rectangles (depicted in green in Fig. 4) is an indicator of the dependence of the characteristics. Unit squares are, therefore, a type of diagram with highly condensed information captured in its graphical appearance.

Tree diagrams can be used as probability trees to model processes, as well as frequency trees to model partitions. An example of the former one are classical multi-level random experiments with events such as the urn problem (drawing balls from an urn with black and white balls), each draw depicted in one level, each level making up a partial event.

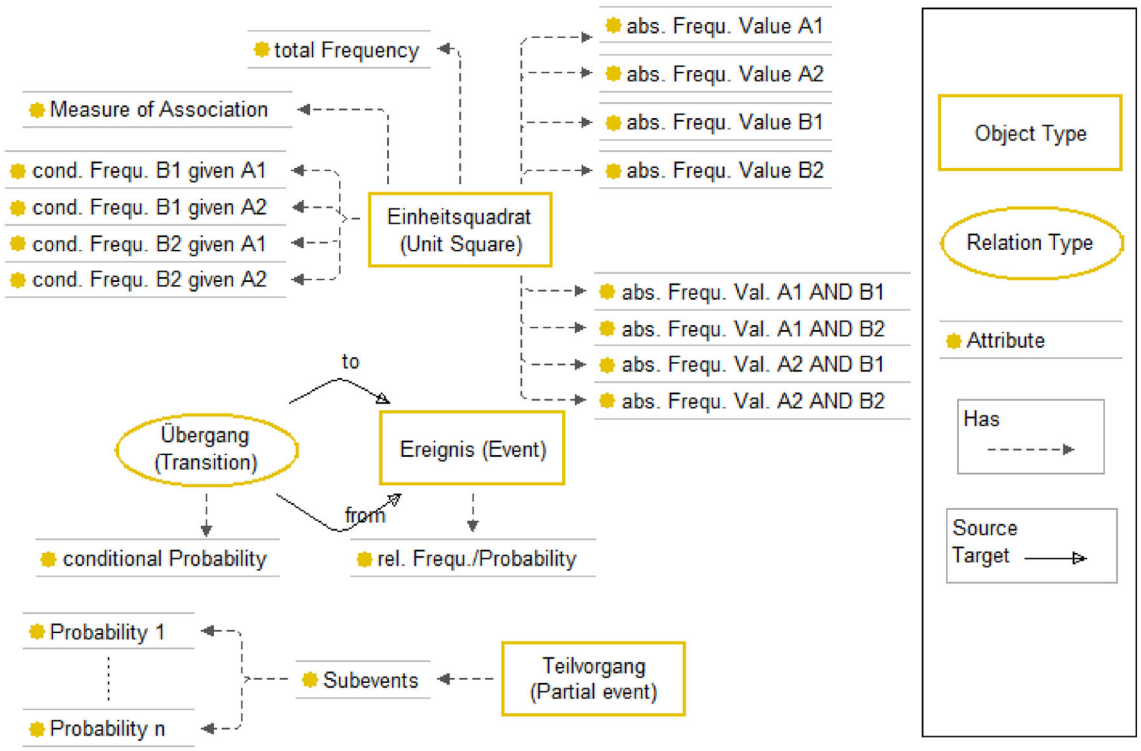


Fig. 3 The metamodel of ProVis

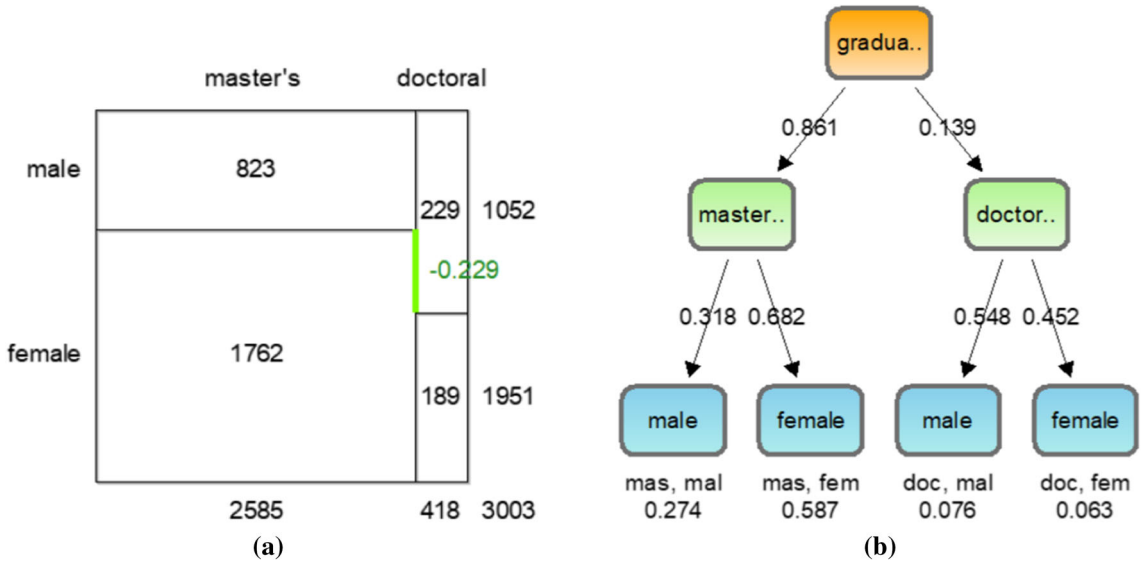


Fig. 4 Number of graduations (master's and doctoral degree) at the University of Vienna in 2019

In probability trees, we depict the probabilities of multi-level events (e.g., drawing a white ball three times in a row) and the conditional probabilities of transitions between two levels (e.g., drawing a white ball after drawing a black one). With knowledge about the partial events of the different levels, the tree can be fully determined.

An example of a frequency tree is the partition of the cohort of graduates at the University of Vienna according to the type of degree and gender, as depicted in Fig. 4b. In a frequency tree, we depict (absolute or relative) frequencies of subgroups of the cohort under study with a concrete characteristic. According to the metamodel, tree diagrams are constituted from events and transitions, which fits the concept of a probability tree. For frequency trees, the notation of events might be unlikely but this kind of trees also serves as a mediator between frequencies and probabilities: The relative frequency of a characteristic can be interpreted as the probability of the event of choosing a member of the cohort with this characteristic in a random sample.

All in all, both model types are highly sophisticated constructs bearing a lot of information with only a few basic input parameters. They are loaded with precisely defined dependencies between their features and graphical appearance. Changing one value automatically implies the adjustment of all other values. Capturing these dependencies and the density of information in a concise manner results in an efficient tool to apply the method. Therefore, it is of high interest to formalize the PROVIS modeling language.

Figure 4 presents the number of master's and doctoral graduations at the University of Vienna in 2019 depending on the gender[1]. This example reveals that although nearly 70% of masters graduates are female (1762 female : 823 male) the rate of females finishing a doctoral program is with around 45% much lower (189 female : 229 male). If we assume that these proportions are similar over the last few years, we can conclude that females finish their academic career after a master's degree more often than males.

In the following, we will outline the formalized modeling language PROVIS concept by concept as appearing in Definition 1. In the full-fledged language, both visualization methods—tree diagrams and unit squares—offer to work with absolute frequencies as well as with probabilities/relative frequencies. In the case study, we restrict unit squares to absolute frequencies and tree diagrams to probabilities/relative frequencies. We do that to reduce the number of attributes but do not lose the expressivity of the formalization.

5.1.1 Object types

From the metamodel, we see that there are three object types **UniSquare (US)**, **Event (E)**, and **PartialEvent (PE)**. These types constitute the types in $\mathcal{S}_O$

$$\mathcal{S}_O = \{\textbf{UnitSquare}, \textbf{Event}, \textbf{PartialEvent}\}.$$

There is no specialization relation between these types: $<_O = \{\}$.

5.1.2 Relation types

The metamodel contains only one relation type **Transition (Tr)**

[1] www.univie.ac.at/en/about-us/at-a-glance/facts-folders/

$\mathcal{S}_R = \{\textbf{Transition}\}$

to connect the nodes in tree diagrams. According to Definition 1 we have to specify two function symbols F_s^{Tr} and F_t^{Tr} indicating source and target types to **Transition**. From the metamodel, we see that this relation type connects elements of type **Event** with other elements of type **Event**

$F_s^{\mathrm{Tr}} : \textbf{Transition} \rightarrow \textbf{Event}$,
$F_t^{\mathrm{Tr}} : \textbf{Transition} \rightarrow \textbf{Event}$.

5.1.3 Attributes

In the following, we will demonstrate how to define attributes and their value domains. We start with single-value attributes and then show how to use product types to define multi-value attributes.

Single-value Attributes A unit square is a graphical representation of a 2×2 contingency table with two characteristics A and B, each of which has two values A_1, A_2, and B_1, B_2, respectively. In Fig. 4, characteristic A is the type of degree, and characteristic B is the gender. A concrete combination of characteristic values is the number of male graduates with a master's degree (A_1B_1). The unit square is built up by the four frequencies, one for each possible combination of attribute values: A_1B_1, A_1B_2, A_2B_1, A_2B_2. Each of them points to an amount, i.e., a natural number including 0. To capture the attribute-value domain and possible attribute values in the language, we have to introduce a value type in $\mathcal{S}_D$ called $\mathbb{N}_0$. The possible values of this type, i.e., constants in $\mathcal{C}$ of type $\mathbb{N}_0$, are 0, 1, 2, 3, ... with their usual semantics.

$\mathbb{N}_0 \in \mathcal{S}_D,\ \mathcal{C} \supset \{0, 1, 2, ...\}$ of type $\mathbb{N}_0$.

With this, we can define the four attributes via function symbols mapping the element of type Unit Square (**US**) to the attribute value in $\mathbb{N}_0$ (attributes *abs. Frequ. Val.* A_i *AND* B_j in the metamodel):

$F_{A_1B_1}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$, $F_{A_2B_1}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$,
$F_{A_1B_2}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$, $F_{A_2B_2}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$.

Also, the absolute numbers of a single value, e.g., the number of all female graduates, as well as the size of the whole cohort, are of interest (attributes *abs. Frequ. Value* A_i and *total Frequency* in the metamodel). We deduce five more function symbols:

$F_{A_1}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$, $F_{A_2}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$,
$F_{B_1}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$, $F_{B_2}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$,
$F_{total}^{abs} : \textbf{US} \rightarrow \mathbb{N}_0$.

For the construction of the visual appearance of a unit square, it is essential to know the conditional frequency of B_1 given A_1 ($B_1|A_1$ in short), etc. (e.g., the percentage of females under all doctoral graduates). This is dependent on the absolute frequencies (the absolute frequency of female doctoral graduates divided by the number of all doctoral graduates) and evaluates to a value between 0 and 1. We specify a new attribute value type **Percentage** and add all real numbers in [0; 1] to the constants in $\mathcal{C}$:

Percentage $\in \mathcal{S}_D,\ \mathcal{C} \supset [0; 1]$ of type **Percentage**.

With this type, we are able to define the required attributes:

$F_{B_1|A_1}^{cond} : \textbf{US} \rightarrow \textbf{Percentage}$, $F_{B_2|A_1}^{cond} : \textbf{US} \rightarrow \textbf{Percentage}$,
$F_{B_1|A_2}^{cond} : \textbf{US} \rightarrow \textbf{Percentage}$, $F_{B_2|A_2}^{cond} : \textbf{US} \rightarrow \textbf{Percentage}$.

Another important feature of a unit square is the measure of association (attribute *Measure of Association* in the metamodel). This value gives a measure for the dependency of the two characteristics A and B and is calculated by subtracting the conditional frequency $B_1|A_2$ from $B_1|A_1$. Graphically it is the difference in the heights of the upper left and upper right rectangles in the square. It evaluates to a value between -1 and 1. A value considerably different from 0 indicates a dependency between the characteristics.

Again, we need a new attribute-value type. We call it $\mathbb{R}$ and give it the usual semantics known from math classes:

$\mathbb{R} \in \mathcal{S}_D,\ \mathcal{C} \supset (-\infty; \infty)$ of type $\mathbb{R}$.

With this value domain, we are now able to define the function symbol for the attribute *measure of association*:

$F^{ass} : \textbf{US} \rightarrow \mathbb{R}$

and thereby complete the formalization of unit squares.

Trees are built up from their single events, depicted by colored rectangles and transitions visualized as arrows between them. The types **Event** and **Transition** both contain only one attribute representing their relative frequency/probability and conditional probability, respectively (in the metamodel attributes *rel. Frequ./Probability* and *conditional Probability*). The corresponding function symbols look as follows:

$F_E^{prob} : \textbf{E} \rightarrow \textbf{Percentage}$, $F_{Tr}^{prob} : \textbf{Tr} \rightarrow \textbf{Percentage}$.

We do not include free text attributes in the formalized language. This is done because the automatic processing of a model requires a machine to understand the semantics of each value. This is only the case when we neatly define the possible values in the attribute-value type. Natural language

descriptions do not qualify for semantic processing. Nevertheless, to cover the usual practices of method engineers, the future research agenda of the proposed formalism METAMORPH also includes free text variables.

Multi-value Attributes The last object type for which we have to define the attributes is the type **PartialEvent**. A partial event is a single-stage event, for example, the second throw, when you roll a dice three times. It can contain several subevents, each of which has a probability (attribute *Subevents* in the metamodel). The partial event of rolling a dice includes following six subevents: *rolling a 1*, *rolling a 2*, ..., *rolling a 6*. Each of these subevents has probability 1/6. This is an example of a multi-value attribute, as a tuple of several values is mapped to the modeling element. For this, we introduce new product types $(\textbf{Percentage})^i$ for an i-valued attribute. A partial event may contain arbitrary many subevents, so we have to build the union of all of these product types and introduce a new attribute value type **Subevents** for it:

$$\textbf{Subevents} = \bigcup_i \textbf{Percentage}^i.$$

This means that the corresponding attribute points to an (arbitrary dimensioned) array of probabilities, i.e., the mentioned subevent of rolling a single dice corresponds to the 6-dimensional tuple $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$ for the probabilities of all six possible outcomes. The attribute results in a function symbol

$$F_{PE}^{sub} : \textbf{PartialEvent} \rightarrow \textbf{Subevents}.$$

5.1.4 Additional symbols

Often we need additional symbols to be able to formulate constraints on our modeling languages. In this case study, we need the symbols of the usual addition, subtraction, division (all function symbols), and order relation on natural numbers to be able to define the dependency between conditional and absolute frequencies, the measure of association, etc.:

$$+_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0 \text{ in } \mathcal{F}, \quad -_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{R} \text{ in } \mathcal{F},$$
$$/_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{R} \text{ in } \mathcal{F}, \quad <_{\mathbb{N}_0} \subset \mathbb{N}_0 \times \mathbb{N}_0 \text{ in } \mathcal{R}.$$

Furthermore, we want a construct for paths on trees, i.e., directed sequences of connected events. To realize this, we introduce a relation symbol:

$$Path \subset \textbf{Event} \times \textbf{Event}.$$

This construct is needed to ensure with constraints the tree like structure of tree diagrams. The usual behavior of paths will be specified in the next section.

5.1.5 Constraints

Modeling languages carry constraints to ensure that only semantically meaningful models are created. These constraints are mostly not depicted in graphical metamodels and have to be added to a language specification by other means. In METAMORPH, they are an integral part of language definition and can be defined in logical sentences using first-order logic.

We split constraints into two categories: restrictive ones that confine the creation, connection, and attribution of models and constructive ones that impose some dependencies between elements and values. The latter can be understood as automatically derived values or instances.

To ease the differentiation between language definition and language instantiation, we use capital letters for the symbols of the language signature and lowercase letters for interpretations of the symbols on the model level.

Constructive Constraints: On tree diagrams, we have two constraints defining how the *Path* relation is deduced from connected elements. To be exact, it is the transitive closure of the relation type **Transition** (each transition makes up a path (1), and if there exists a path from x to y and one from y to z, then also from x to z (2)) .

$$\forall t \in \textbf{Transition}, x, y \in \textbf{Event}$$
$$(f_s^{\mathrm{Tr}}(t) = x \wedge f_t^{\mathrm{Tr}}(t) = y \implies Path(x, y)) \tag{1}$$
$$\forall x, y, z \in \textbf{Event}$$
$$(Path(x, y) \wedge Path(y, z) \implies Path(x, z)) \tag{2}$$

A third constraint is given by the multiplication rule, also called the path rule in math class. This rule states how the probability of a compound event x can be calculated from the preceding transition and event, t and $f_s^{\mathrm{Tr}}(t)$, in the tree diagram (e.g., the percentage of the male doctoral graduates derived from the conditional frequency of the incoming transition and the relative frequency of all doctoral graduates).

$$\forall x \in \mathbf{E}, t \in \mathbf{Tr}\ (f_t^{\mathrm{Tr}}(t) = x \implies$$
$$f_E^{prob}(x) = f_{Tr}^{prob}(t) \cdot f_E^{prob}(f_s^{\mathrm{Tr}}(t))) \tag{3}$$

On unit squares, we obtain a couple of calculation rules. The absolute frequency of characteristic values A_1, A_2, B_1, B_2 are fully dependent on the frequencies of value combinations, e.g., the number of all doctoral graduates A_2 is the sum of male doctoral graduates A_2B_1 and female doctoral graduates A_2B_2.

$$\forall x \in \mathbf{US}\ (f_{A_1}^{abs}(x) = f_{A_1B_1}^{abs}(x) + f_{A_1B_2}^{abs}(x)) \tag{4}$$
$$\forall x \in \mathbf{US}\ (f_{A_2}^{abs}(x) = f_{A_2B_1}^{abs}(x) + f_{A_2B_2}^{abs}(x)) \tag{5}$$
$$\forall x \in \mathbf{US}\ (f_{B_1}^{abs}(x) = f_{A_1B_1}^{abs}(x) + f_{A_2B_1}^{abs}(x)) \tag{6}$$

$$\forall x \in \mathbf{US}\ (f^{abs}_{B_2}(x) = f^{abs}_{A_1B_2}(x) + f^{abs}_{A_2B_2}(x)) \tag{7}$$

Also, the frequency of the whole cohort can be calculated by summing up the numbers of all subgroups.

$$\forall x \in \mathbf{US}\ (f^{abs}_{total}(x) = f^{abs}_{A_1B_1}(x) + f^{abs}_{A_1B_2}(x) + f^{abs}_{A_2B_1}(x) + f^{abs}_{A_2B_2}(x)) \tag{8}$$

Four more constructive constraints are given by the conditional frequencies $B_i|A_j$, which also can be automatically calculated according to the definition $f(X|Y) = f(XY)/f(Y)$.

$$\forall x \in \mathbf{US}(f^{cond}_{B_1|A_1}(x) = f^{abs}_{A_1B_1}(x)/f^{abs}_{A_1}(x)) \tag{9}$$

$$\forall x \in \mathbf{US}(f^{cond}_{B_2|A_1}(x) = f^{abs}_{A_1B_2}(x)/f^{abs}_{A_1}(x)) \tag{10}$$

$$\forall x \in \mathbf{US}(f^{cond}_{B_1|A_2}(x) = f^{abs}_{A_2B_1}(x)/f^{abs}_{A_2}(x)) \tag{11}$$

$$\forall x \in \mathbf{US}(f^{cond}_{B_2|A_2}(x) = f^{abs}_{A_2B_2}(x)/f^{abs}_{A_2}(x)) \tag{12}$$

Finally, the measure of association f^{ass} can be calculated. It is given by the difference in the heights of the upper left and right rectangle in the unit square

$$\forall x \in \mathbf{US}(f^{ass}(x) = f^{cond}_{B_1|A_1}(x) - f^{cond}_{B_1|A_2}(x)) \tag{13}$$

Restrictive Constraints: The most prominent restrictive constraints are cardinality constraints. Tree diagrams allow at most one incoming relation to an element of type **Event**. This is specified in Equation (14). Similar to that, constraints of higher cardinality can be defined. Furthermore, a tree cannot contain circles. This is specified using the *Path* relation (15).

$$\forall x, y \in \mathbf{Transition}(f^{\mathrm{Tr}}_t(x) = f^{\mathrm{Tr}}_t(y) \implies x = y) \tag{14}$$

$$\forall x \in \mathbf{Event}(\neg Path(x, x)) \tag{15}$$

To make sure that the *Path* relation exclusively reflects the concatenated relation elements of type **Transition**, we need a further constraint. For brevity we use the abbreviation xty for relation t of type $\mathbf{T}$, x being the source $\mathtt{F}^{\mathbf{T}}_s(r)$ and y being the target $\mathtt{F}^{\mathbf{T}}_t(r)$ of t):

$$\forall x, y \in \mathbf{E}\ \exists z \in \mathbf{E}, t \in \mathbf{Tr}\ (Path(x, y) \implies xty \vee (xtz \wedge Path(z, y))) \tag{16}$$

5.1.6 The complete formalized language ProVis

In summary, we collect all the symbols constituting the language ProVis:

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

$\mathcal{S}_O = \{\mathbf{UnitSquare}, \mathbf{Event}, \mathbf{PartialEvent}\}, <_O = \{\}$

$\mathcal{S}_R = \{\mathbf{Transition}\}$,

$\mathcal{S}_D = \{\mathbb{N}_0,\ \mathbb{R},\ \mathbf{Percentage},\ \mathbf{Subevents} = \bigcup_i \mathbf{Percentage}^i\}$,

$\mathcal{F} = \{F^{\mathrm{Tr}}_s : \mathbf{Transition} \to \mathbf{Event}$,

$F^{\mathrm{Tr}}_t : \mathbf{Transition} \to \mathbf{Event}$,

$F^{abs}_{A_1B_1} : \mathbf{US} \to \mathbb{N}_0$, $F^{abs}_{A_2B_1} : \mathbf{US} \to \mathbb{N}_0$,

$F^{abs}_{A_1B_2} : \mathbf{US} \to \mathbb{N}_0$, $F^{abs}_{A_2B_2} : \mathbf{US} \to \mathbb{N}_0$,

$F^{abs}_{A_1} : \mathbf{US} \to \mathbb{N}_0$, $F^{abs}_{A_2} : \mathbf{US} \to \mathbb{N}_0$,

$F^{abs}_{B_1} : \mathbf{US} \to \mathbb{N}_0$, $F^{abs}_{B_2} : \mathbf{US} \to \mathbb{N}_0$,

$F^{abs}_{total} : \mathbf{US} \to \mathbb{N}_0$, $F^{ass} : \mathbf{US} \to \mathbb{R}$,

$F^{cond}_{B_1|A_1} : \mathbf{US} \to \mathbf{Percentage}$, $F^{cond}_{B_2|A_1} : \mathbf{US} \to \mathbf{Percentage}$,

$F^{cond}_{B_1|A_2} : \mathbf{US} \to \mathbf{Percentage}$, $F^{cond}_{B_2|A_2} : \mathbf{US} \to \mathbf{Percentage}$,

$F^{prob}_E : \mathbf{Event} \to \mathbf{Percentage}$,

$F^{prob}_{Tr} : \mathbf{Transition} \to \mathbf{Percentage}$,

$F^{sub}_{PE} : \mathbf{PartialEvent} \to \mathbf{Subevents}$,

$+_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, $-_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$,

$/_{\mathbb{N}_0} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}\}$,

$\mathcal{R} = \{Path \subset \mathbf{Event} \times \mathbf{Event}, <_{\mathbb{N}_0} \subset \mathbb{N}_0 \times \mathbb{N}_0\}$,

$\mathcal{C} = \{0, 1, 2, ...\}$ of type $\mathbb{N}_0 \cup [0; 1]$ of type **Percentage** $\cup(-\infty; \infty)$ of type $\mathbb{R}$.

Furthermore we accumulated 16 constraints:

$\forall t \in \mathbf{Transition}, x, y \in \mathbf{Event}$

$$(f^{\mathrm{Tr}}_s(t) = x \wedge f^{\mathrm{Tr}}_t(t) = y \implies Path(x, y)) \tag{1}$$

$\forall x, y, z \in \mathbf{Event}$

$$(Path(x, y) \wedge Path(y, z) \implies Path(x, z)) \tag{2}$$

$$\forall x \in \mathbf{E}, t \in \mathbf{Tr}\ (f^{\mathrm{Tr}}_t(t) = x \implies f^{prob}_E(x) = f^{prob}_{Tr}(t) \cdot f^{prob}_E(f^{\mathrm{Tr}}_s(t))) \tag{3}$$

$$\forall x \in \mathbf{US}\ (f^{abs}_{A_1}(x) = f^{abs}_{A_1B_1}(x) + f^{abs}_{A_1B_2}(x)) \tag{4}$$

$$\forall x \in \mathbf{US}\ (f^{abs}_{A_2}(x) = f^{abs}_{A_2B_1}(x) + f^{abs}_{A_2B_2}(x)) \tag{5}$$

$$\forall x \in \mathbf{US}\ (f^{abs}_{B_1}(x) = f^{abs}_{A_1B_1}(x) + f^{abs}_{A_2B_1}(x)) \tag{6}$$

$$\forall x \in \mathbf{US}\ (f^{abs}_{B_2}(x) = f^{abs}_{A_1B_2}(x) + f^{abs}_{A_2B_2}(x)) \tag{7}$$

$$\forall x \in \mathbf{US}\ (f^{abs}_{total}(x) = f^{abs}_{A_1B_1}(x) + f^{abs}_{A_1B_2}(x) + f^{abs}_{A_2B_1}(x) + f^{abs}_{A_2B_2}(x)) \tag{8}$$

$$\forall x \in \mathbf{US}(f^{cond}_{B_1|A_1}(x) = f^{abs}_{A_1B_1}(x)/f^{abs}_{A_1}(x)) \tag{9}$$

$$\forall x \in \mathbf{US}(f^{cond}_{B_2|A_1}(x) = f^{abs}_{A_1B_2}(x)/f^{abs}_{A_1}(x)) \tag{10}$$

$$\forall x \in \mathbf{US}(f^{cond}_{B_1|A_2}(x) = f^{abs}_{A_2B_1}(x)/f^{abs}_{A_2}(x)) \tag{11}$$

$$\forall x \in \mathbf{US}(f^{cond}_{B_2|A_2}(x) = f^{abs}_{A_2B_2}(x)/f^{abs}_{A_2}(x)) \tag{12}$$

$$\forall x \in \mathbf{US}(f^{ass}(x) = f^{cond}_{B_1|A_1}(x) - f^{cond}_{B_1|A_2}(x)) \tag{13}$$

$$\forall x, y \in \mathbf{Transition}(f^{\mathrm{Tr}}_t(x) = f^{\mathrm{Tr}}_t(y) \implies x = y) \tag{14}$$

$$\forall x \in \mathbf{Event}(\neg Path(x, x)) \tag{15}$$

$$\forall x, y \in \mathbf{E}\ \exists z \in \mathbf{E}, t \in \mathbf{Tr}\ (Path(x, y) \implies xty \vee (xtz \wedge Path(z, y))) \tag{16}$$

We see here that the whole language PROVIS with its numerous attributes and dependencies can be comprehensively captured in a highly concise manner in a couple of lines. Nevertheless, it provides the full power of deducing information about conditional frequencies and dependencies of characteristics, as well as probabilities and frequencies of compound events. The formalized language is, therefore, an extremely precise, concise, and efficient tool for the deployment of the modeling method and exceeds the use of a natural language specification in these terms.

5.1.7 Formalizing graphical models

Female graduates at the University of Vienna: We formalize the example presented in Fig. 4. First of all, we define the universes of elements of all types. There is one element *grads* of type **Unit Square** and seven elements of type **Event** constituting the tree diagram. For reasons of uniqueness, we renamed the events in the last row of the tree diagram.

$$\mathcal{U}_{US} = \{grads\}, \mathcal{U}_{PE} = \{\},$$
$$\mathcal{U}_{E} = \{graduates, masters, doctors, mMasters, fMasters, mDoctors, fDoctors\}.$$

The tree furthermore contains six relation elements of type **Transition**:

$$\mathcal{U}_{Tr} = \{t_1, t_2, t_3, t_4, t_5, t_6\}.$$

For the six relation elements of type **Transition** we have to concretize the source and target values:

$$f_s^{\mathrm{Tr}}(t_1) = graduates,\ f_t^{\mathrm{Tr}}(t_1) = masters,$$
$$f_s^{\mathrm{Tr}}(t_2) = graduates,\ f_t^{\mathrm{Tr}}(t_2) = doctors,$$
$$f_s^{\mathrm{Tr}}(t_3) = masters,\ f_t^{\mathrm{Tr}}(t_3) = mMasters,$$
$$f_s^{\mathrm{Tr}}(t_4) = masters,\ f_t^{\mathrm{Tr}}(t_3) = fMasters,$$
$$f_s^{\mathrm{Tr}}(t_5) = doctors,\ f_t^{\mathrm{Tr}}(t_5) = mDoctors,$$
$$f_s^{\mathrm{Tr}}(t_6) = doctors,\ f_t^{\mathrm{Tr}}(t_6) = fDoctors.$$

The base attribute values, i.e., the conditional frequencies of transitions, needed in the tree diagram in Fig. 4 right are the following:

$$f_E^{prob}(graduates) = 1,\ f_{Tr}^{prob}(t_1) = 0.861,$$
$$f_{Tr}^{prob}(t_2) = 0.139,\ f_{Tr}^{prob}(t_3) = 0.318,$$
$$f_{Tr}^{prob}(t_4) = 0.682,\ f_{Tr}^{prob}(t_5) = 0.548,\ f_{Tr}^{prob}(t_6) = 0.452$$

The multiplication rule (constraint (3)) allows for the calculation of the relative frequencies of the events using the conditional frequencies of the transitions ($f_E^{prob}(f_t^{\mathrm{Tr}}(t)) = f_{TR}^{prob}(t) \cdot f_E^{prob}(f_s^{\mathrm{Tr}}(t))$):

$$f_E^{prob}(masters) = 1 \cdot 0.861 = 0.861,$$
$$f_E^{prob}(doctors) = 1 \cdot 0.139 = 0.139,$$
$$f_E^{prob}(mMasters) = 0.861 \cdot 0.318 \approx 0.274,$$
$$f_E^{prob}(fMasters) = 0.861 \cdot 0.682 \approx 0.587,$$
$$f_E^{prob}(mDoctors) = 0.139 \cdot 0.548 \approx 0.076,$$
$$f_E^{prob}(fDoctors) = 0.139 \cdot 0.452 \approx 0.063.$$

The four base attribute values of the given unit square *grads* look as follows (in this example A_1B_1 = # male masters graduates, A_2B_1 = # male doctoral graduates, A_1B_2 = # female masters graduates, A_2B_2 = # female doctoral graduates):

$$f_{A_1B_1}^{abs}(grads) = 823,\ f_{A_2B_1}^{abs}(grads) = 229,$$
$$f_{A_1B_2}^{abs}(grads) = 1762,\ f_{A_2B_2}^{abs}(grads) = 189.$$

With the constructive constraints from Sect. 5.1.5 we derive automatically the frequencies of the different characteristics and the frequency of the whole cohort (in this example A_1= # master's degrees, A_2= # doctoral degrees, B_1= # male graduates, B_2= # female graduates):

$$f_{A_1}^{abs}(grads) = 823 + 1762 = 2585,$$
$$f_{A_2}^{abs}(grads) = 229 + 189 = 418,$$
$$f_{B_1}^{abs}(grads) = 823 + 229 = 1052,$$
$$f_{B_2}^{abs}(grads) = 1762 + 189 = 1951.,$$
$$f_{total}^{abs}(grads) = 823 + 229 + 1762 + 189 = 3003.$$

For the conditional probabilities $B_i|A_j$ we calculate with constraint (9)–(13) ($f_{B_i|A_j}^{cond}(x) = f_{A_jB_i}^{abs}(x)/f_{A_j}^{abs}(x)$):

$$f_{B_1|A_1}^{cond}(grads) = 823/2585 = 0.318,$$
$$f_{B_1|A_2}^{cond}(grads) = 229/418 = 0.548,$$
$$f_{B_2|A_1}^{cond}(grads) = 1762/2585 = 0.682,$$
$$f_{B_2|A_2}^{cond}(grads) = 159/418 = 0.452.$$

Also, the measure of association can be deduced automatically:

$$f^{ass}(grads) = 0.318 - 0.548 = -0.23.$$

Its value differs considerably from 0 and therefore indicates a dependency between gender and graduation.

This concludes the full formalization of the example from Fig. 4.

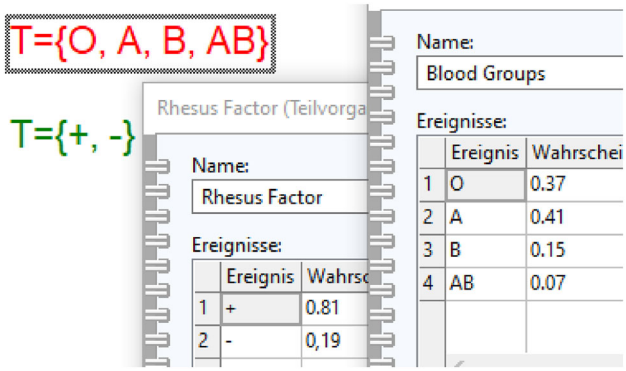


Fig. 5 Partial events concerning blood groups and rhesus factor

Distribution of Blood Groups in Austria: We also want to demonstrate the formalization of partial events. This is done using the example of the distribution of blood groups and rhesus factor in Austria[2] shown in Fig. 5. There, we see two examples of partial events representing the probabilities of having blood group O, A, B, or AB and the probabilities of being of positive- or negative rhesus factor.

The universe of partial events therefore is

$$\mathcal{U}_{PE} = \{bloodGroups, rhesusFactor\}.$$

The multi-value attribute

$$F_{PE}^{sub} : \textbf{PartialEvent} \rightarrow \textbf{Subevents} = \bigcup_i \textbf{Percentage}^i$$

maps the two elements to the two tuples:

$$f_{PE}^{sub}(bloodGroups) = (0.37, 0.41, 0.15, 0.07)$$

for the relative frequencies of the four blood groups and

$$f_{PE}^{sub}(rhesusFactor) = (0.81, 0.19)$$

for the relative frequencies of positive and negative rhesus factor.

From these details and the fact that the two partial events are independent of each other, we can construct a tree diagram depicted in Fig. 6. With the postulated constraints of the multiplication rule (3), we can furthermore calculate the probabilities of the compound event, having blood group A with positive rhesus factor, etc., revealing that the most frequent blood combination in Austria is A+ (33.2%) whereas the rarest one is AB− (1.3%). The formalization of the tree diagram in Fig. 6 is not outlined here.

[2] www.gesundheit.gv.at/labor/laborwerte/blutgruppenserologie-transfusion/blutgruppenuntersuchung1-kh

5.2 Formalizing the Entity-Relationship modeling language

To show the domain-independence of MetaMorph and complete the demonstration of all the concepts comprised in the formalism we complement the case study on ProVis with the application of MetaMorph to the well-known Entity-Relationship (ER) modeling language. Initially introduced by Chen in 1976 [10] nowadays, there are several variants of the language with slight semantic and notational differences, e.g., different handling of non-binary relationships. We follow the introduction to ER from [51, 7.5] using the famous Chen-notation and derive the metamodel of the ER language depicted in Fig. 7. The Chen-notation includes an additional modeling concept for relationships (the diamonds), while other notation systems depict relationships only via arrows. This led us to design the metamodel with a supertype for Entities and Relationships to facilitate the definition of connections between them. Similar to ProVis the ER language is characterized by its simplicity. It contains only a few concepts by offering a high expressivity in the resulting models.

In ER models, the concepts entity ("a "thing" which can be distinctly identified" [10, p.10]) and relationship ("an association among entities" [10, p.10]) are used to model data structures and express the associations and dependencies between entities. Thereby, relationships usually connect two entities, but also non-binary relationships are permitted. Both concepts can be further enriched with information via attributes that might function as unique identifiers for an entity. Furthermore, cardinality constraints can be used to refine the numerical relationship between entities, e.g., one-to-one, one-to-many, etc. These concepts build the core of the ER language. Furthermore, the advanced concepts of weak entities and composite attributes are introduced to meet the subtle requirements of data modeling.

Weak entities are a special subtype of entities that do not contain sufficient information for a unique identifier. They are dependent on related entities, such that the combination of objects allows for the unique identification of the weak entity. The related entity is called the identifying or owning entity. Weak entities can be connected to more than one other object but need at least one connection to an identifying entity. In a model, the relationship object connecting the weak entity with the identifying entity is notationally marked. According to [51] the participation of the weak entity in the identifying relation must be total, i.e., the maximum and minimum cardinality must be set to 1. Attributes of weak entities can play the role of a discriminator, i.e., a value that, in combination with the identifier of the owning entity, uniquely identifies the weak entity.

Composite attributes allow for the handling of attributes with complex structures, e.g., addresses. An address might

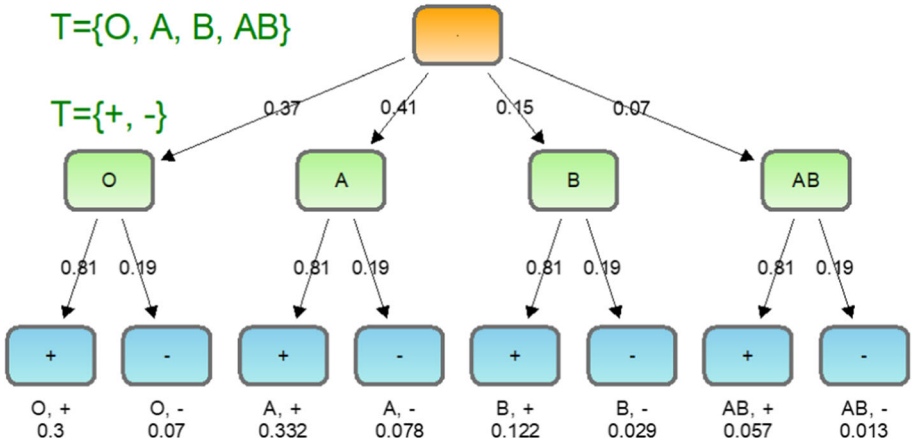


Fig. 6 Distribution of blood groups and rhesus factor in Austria

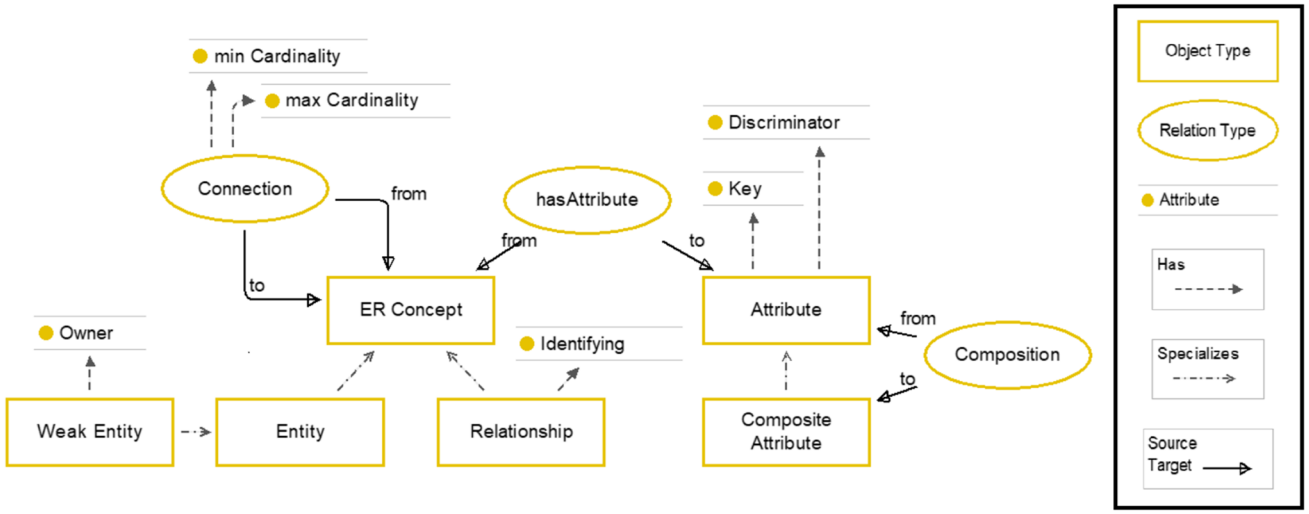


Fig. 7 Metamodel of the ER language according to [51]

be captured in a composite attribute constituted of the single attributes *street*, *city*, *state*, and *zip-code*.

5.2.1 Object types

The object types **Entity** (**En**), **Relationship** (**Re**), and **Attribute** (**Attr**) represent the core of the ER modeling language. Furthermore, we include an abstract supertype **ER Concept** for **Entity** and **Relationship** to allow for the omni-directional connection of instances of the two subtypes and the relation to attributes of both. We also include the concepts of **Weak Entity** (**WE**), a subtype of **Entity**, and **Composite Attribute** (**CA**), subtype of **Attribute**. Weak entities are designed as their own object type to admit specific attributes not relevant for entities in general. Composite attributes are designed as their own object type so that we can define the **Composition** relation between them.

These types constitute the sorts in $\mathcal{S}_O$

$\mathcal{S}_O = \{$**ER Concept (ERC)**, **Entity (En)**, **Weak Entity (WE)**, **Relationship (Re)**, **Attribute (Attr)**, **Composite Attribute (CA)**$\}$

We outlined several specialization dependencies between those types:

$<_O = \{$(**Entity, ER Concept**), (**Relationship, ER Concept**), (**Weak Entity, Entity**), (**Composite Attribute, Attribute**)$\}$

5.2.2 Relation types

For the relations of the main concepts, we introduce relation types **Connection (Con)** and **hasAttribute (HA)** to link entities with relationships and attributes with ER-elements, respectively.

$\mathcal{S}_R \supseteq \{$**Connection (Con)**, **HasAttribute (HA)**$\}$

To allow for the linking of entities and relationships in both directions we define **Con** as a relation type from elements of type **ER Concept** to **ER Concept**.

F_s^{Con} : **Connection** → **ER Concept**,
F_t^{Con} : **Connection** → **ER Concept**.

With this construction, it is irrelevant if the entity is source or target or if the relationship is target or source of the **Con** element.

In some notation conventions, some cardinality constraints (e.g., one-to-one) are also reflected in the "direction" of the connection element, but this interferes with the (domain independent) conception of the direction of a relation being the order of drawing (from source to target). This conception of direction does not allow for a change caused by a change in attribute values. A precise way of capturing cardinality mappings is the definition of minimum and maximum cardinality via attributes, see 5.2.3.

With the relation type **HA** we intend to link entities or relationships to their attributes:

F_s^{HA} : **HasAttribute** → **ER Concept**,
F_t^{HA} : **HasAttribute** → **Attribute**.

The additional possibility of collating single attributes to a composite attribute further requires an additional relation type called **Composition (Comp)**:

$\mathcal{S}_R \supseteq$ {**Composition (Comp)**},
F_s^{Comp} : **Composition** → **Attribute**,
F_t^{Comp} : **Composition** → **Composite Attribute**.

5.2.3 Attributes

In the following, we will outline the attributes and their domains of the ER modeling language.

Single-value Attributes The ER language comprises several attributes for specifying cardinalities or determining the features of concrete concepts, e.g., attributes being the primary key. Similar to the ProVis language, also in ER models, the relation elements do have attributes, which affirms the design of MetaMorph defining relation types as their own entities instead of pairs of object elements.

To be able to specify minimal cardinalities, we need a domain containing all natural numbers starting from 0. For the max cardinalities, we additionally admit ∞ to indicate no upper boundary. We, therefore, get a value domain type:

$\mathcal{S}_D \supseteq \{\mathbb{N}_\infty\}$

with constants:

$\mathcal{C} \supseteq \{0, 1, 2, ..., \infty\}$ of type $\mathbb{N}_\infty$.

The cardinality attributes for the relation type **Connection** can now be specified:

F_{minCar}^{Con} : **Connection** → $\mathbb{N}_\infty$,
F_{maxCar}^{Con} : **Connection** → $\mathbb{N}_\infty$.

We will later constrain the minimal cardinality to be unequal to ∞.

The feature of attributes being primary key, discriminator of a weak entity, or multi-valued, and the feature of relationships being identifying for a weak entity are designed as boolean values with constants *true* and *false*:

$\mathcal{S}_D \supseteq$ {**Bool**}
$\mathcal{C} \supseteq \{true(t), false(f)\}$ of type **Bool**.

With this domain, we are able to define the characteristics mentioned earlier via attributes assigning a boolean value *true* (the Attribute element is a discriminator for the weak entity) or *false* (the Attribute element does not have the role of a discriminator for the connected weak entity) to an element of type **Attr** or **Re**.

F_{Key}^{Attr} : **Attribute** → **Bool**,
$F_{Discrim}^{Attr}$: **Attribute** → **Bool**,
F_{Ident}^{Re} : **Relationship** → **Bool**.

Multi-value Attributes We furthermore define an attribute identifying the owning entity set to a weak entity. According to [51] this can also be another weak entity and need not be a single element. Therefore, the owning entity set is a subset of all elements of type **En**. To capture this in a value domain, we have to work with the powerset of entities named **Entities**:

$\mathcal{S}_D \supseteq$ {**Entities** = $\wp$(**Entity**)}.

The reference of a weak entity to its owning entity or set of entities is therefore captured in the attribute pointing to a (potentially singleton) set of entities:

F_{owner}^{WE} : **Weak Entity** → **Entities**.

This owning entity is uniquely defined by the behavior of neighboured relations of type **Con**, and objects of type **Re** and **En**.

5.2.4 Additional symbols

For the definition of constraints, we have to introduce two relation symbols well-known in mathematics. The first one is the usual order relation on natural numbers to ensure the proper behavior of the cardinality attributes. The second one admits the containment check of an entity in a set of entities, i.e., an element in the powerset $\mathbf{Entities} = \wp(\mathbf{Entity})$.

$$\mathcal{R} = \{\leq_{\mathbb{N}_\infty} \subseteq \mathbb{N}_\infty \times \mathbb{N}_\infty,\ \in_{En} \subseteq \mathbf{En} \times \wp(\mathbf{En})\},$$

5.2.5 Constraints

Similar to the case study on PROVIS, the constraints of the ER modeling language are not contained in the graphical metamodel in Fig. 7. The constraints in the following are deduced from the textual description given in [51] without the intention to evaluate correctness or completeness. Our objective is to demonstrate the formalization of a given specification and to point to possible inconsistencies or shortcomings that might turn up during formalization. Again we divide the constraints into restrictive and constructive ones, whereby, in this case, the restrictive ones are the majority.

Restrictive Constraints: Due to the design decision of the abstract class **ER Concept** we need several constraints ensuring **ERC** to be abstract (17) and to ensure alternation of **En** and **Re** as endpoints of relations of type **Con** (18–19).

$$\forall x \in \mathbf{ERC}\ \exists e \in \mathbf{En}, r \in \mathbf{Re}\ (x = e \vee x = r) \tag{17}$$

$$\nexists e, e' \in \mathbf{En}, c \in \mathbf{Con}\ (F_s^{Con}(c) = e \wedge F_t^{Con}(c) = e') \tag{18}$$

$$\nexists r, r' \in \mathbf{Re}, c \in \mathbf{Con}\ (f_s^{Con}(c) = r \wedge f_t^{Con}(c) = r') \tag{19}$$

The relation type **Connection** linking entities with relationships carries the information about minimum and maximum cardinalities in two attributes. Thereby, the min value must not be equal to ∞, and the max value must not be smaller than the min value:

$$\forall c \in \mathbf{Con}(f_{minCar}^{Con}(c) \neq \infty) \tag{20}$$

$$\forall c \in \mathbf{Con}(f_{minCar}^{Con}(c) \leq_{\mathbb{N}_\infty} f_{maxCar}^{Con}(c)) \tag{21}$$

An element r of type **Relationship** must be related to at least two elements of type **Entity**. This is equivalent to two relation objects of type **Connection** linked to r, as these connections necessarily link r to entities. It suffices to define this constraint for outgoing connections as we define **Con** to be omnidirectional, see constraint (33):

$$\forall r \in \mathbf{Re}\ \exists c, c' \in \mathbf{Con}\ (f_s^{\mathrm{Con}}(c) = r \wedge f_s^{\mathrm{Con}}(c') = r \wedge c \neq c') \tag{22}$$

By allowing more than two relation elements of type **Con** the ER language allows for modeling non-binary relationships. Unfortunately, in [51] cardinality mappings in this complex case are only discussed in terms of the "direction" of a **Con** element (in the sense of a notational convention, see 5.2.2). This lacks a precise definition of the semantics of explicit min and max constraint values as well as concrete requirements regarding semantically and syntactically valid models. Therefore, we cannot postulate adequate constraints to ensure the proper behavior of ER models with non-binary relationships. This shows that the formalization process can reveal conceptual shortcomings of a non-formal natural language specification.

An element of type **Attribute** must be connected to an entity or relationship element (i.e., an element of type **ER Concept**) or to a composite attribute but not to both at the same time. It is either target of a relation object of type **HasAttribute** or of an relation object of type **Composition**.

$$\forall a \in \mathbf{Attr}\ \exists h \in \mathbf{HA}\ , c \in \mathbf{Comp}\ ((f_t^{\mathrm{HA}}(h) = a \vee f_s^{\mathrm{Comp}}(c) = a) \wedge \neg(f_t^{\mathrm{HA}}(h) = a \wedge f_t^{\mathrm{Comp}}(c) = a)) \tag{23}$$

Furthermore, an element of type **Attribute** has at most one relation of type **Comp** and at most one relation of type **HA**.

$$\forall a \in \mathbf{Attr}\ h, h' \in \mathbf{HA}\ (f_t^{\mathrm{HA}}(h) = a \wedge f_t^{\mathrm{HA}}(h') = a \implies h = h') \tag{24}$$

$$\forall a \in \mathbf{Attr}\ c, c' \in \mathbf{Comp}\ (f_s^{\mathrm{Comp}}(c) = a \wedge f_s^{\mathrm{Comp}}(c') = a \implies c = c') \tag{25}$$

Each composite attribute is related to at least one attribute, i.e., has at least one incoming relation object of type **Composition**:

$$\forall x \in \mathbf{CA}\ \exists c \in \mathbf{Comp}\ (f_t^{\mathrm{HA}}(c) = x) \tag{26}$$

To ensure the proper behavior of weak entities, we need a couple of restrictions on models. To shorten the following constraints we introduce the abbreviation ecr for e an entity, c a connection, r a relationship, and c connecting e and r: $ecr := (f_s^{\mathrm{Con}}(c) = e \wedge f_t^{\mathrm{Con}}(c) = r)$.

All weak entities are connected to at least one identifying relationship, and its participation in this relationship is total, i.e., the min and max cardinality constraints of the connection are both equal to 1.

$$\forall w \in \mathbf{WE}\ \exists r \in \mathbf{Re}\ , c \in \mathbf{Con}\ (wcr \wedge f_{Ident}^{Re}(r) = true \wedge f_{minCar}^{Con}(c) = 1 \wedge f_{maxCar}^{Con}(c) = 1) \tag{27}$$

At the same time, a relationship can only be identifying when it is connected to a weak entity.

$$\forall r \in \mathbf{Re}\ \exists w \in \mathbf{WE}\ , c \in \mathbf{Con}$$
$$(f^{Re}_{Ident}(r) = t \implies wcr) \tag{28}$$

We also have to restrict the attribute of owning entities af a weak entity w. If an element belongs to the set of owning elements of w it must be connected to w via an identifying relation.

$$\forall w \in \mathbf{WE}\ , e \in \mathbf{En} \exists r \in \mathbf{Re}\ , c \in \mathbf{Con}$$
$$(e \in_{En} f^{WE}_{owner}(w) \implies wcr \wedge f^{Re}_{Ident}(r) = t) \tag{29}$$

For attributes, we need the following constraints: An attribute can only be a discriminator when it is connected to a weak entity (30), it cannot be key and discriminator at the same time (31), and we do not admit situations, in which a weak entity has a key attribute (32).

$$\forall a \in \mathbf{Attr}\ \exists w \in \mathbf{WE}\ , h \in \mathbf{HA}\ (f^{Attr}_{Discrim}(a) = t \implies$$
$$(f^{\mathrm{HA}}_{s}(h) = w \wedge f^{\mathrm{HA}}_{t}(h) = a)) \tag{30}$$
$$\forall a \in \mathbf{Attr}\ (\neg(f^{Attr}_{Key}(a) = t \wedge f^{Attr}_{Discrim}(a) = t)) \tag{31}$$
$$\nexists w \in \mathbf{WE}\ , a \in \mathbf{Attr}\ , h \in \mathbf{HA}$$
$$(f^{\mathrm{HA}}_{s}(h) = w \wedge f^{\mathrm{HA}}_{t}(h) = a \wedge f^{Attr}_{Key}(a) = t) \tag{32}$$

Constructive Constraints: To factually make the **Connection** relation type omnidirectional, we add to each relation object r the inverse relation object pointing from the target of r to the source of r:

$$\forall x, y \in \mathbf{ERC}\ c \in \mathbf{Con}\ \exists c' \in \mathbf{Con}$$
$$((f^{\mathrm{Con}}_{s}(c) = x \wedge f^{\mathrm{Con}}_{t}(c) = y) \implies$$
$$(f^{\mathrm{Con}}_{s}(c') = y \wedge f^{\mathrm{Con}}_{t}(c') = x)) \tag{33}$$

The reverse connection must expose the same cardinalities:

$$\forall c, c' \in \mathbf{Con}\ ((f^{\mathrm{Con}}_{s}(c) = f^{\mathrm{Con}}_{t}(c') \wedge f^{\mathrm{Con}}_{t}(c) = f^{\mathrm{Con}}_{s}(c'))$$
$$\implies (f^{Con}_{minCar}(c) = f^{Con}_{minCar}(c')$$
$$\wedge f^{Con}_{maxCar}(c) = f^{Con}_{maxCar}(c'))) \tag{34}$$

We defined an attribute F^{WE}_{owner} mapping a weak entity to the set of identifying or owning entities. This information can be automatically derived from the constellation of relationships and entities. Thus, we are able to deduce this set of entities automatically via a constructive constraint:

$$\forall w \in \mathbf{WE}\ , r \in \mathbf{Re}\ , e \in \mathbf{En}\ , c, c' \in \mathbf{Con}$$
$$((f^{Re}_{Ident}(r) = true \wedge wcr \wedge ec'r \wedge w \neq e)$$
$$\implies e \in_{En} f^{WE}_{owner}(w)) \tag{35}$$

5.2.6 The complete formalized ER language

In summary, we collect all the symbols constituting the ER language.

$\Sigma = \{\mathcal{S}, \mathcal{F}, \mathcal{R}, \mathcal{C}\}, \mathcal{S} = \mathcal{S}_O \cup \mathcal{S}_R \cup \mathcal{S}_D$
$\mathcal{S}_O = \{$**ER Concept (ERC)**, **Entity (En)**,
Weak Entity(WE), **Relationship (Re)**,
Attribute (Attr), **Composite Attribute (CA)**$\}$,
$<_O = \{$(**Entity, ER Concept**),
(**Relationship, ER Concept**), (**Weak Entity, Entity**),
(**Composite Attribute, Attribute**)$\}$
$\mathcal{S}_R = \{$**Connection (Con)**, **HasAttribute (HA)**,
Composition (Comp)$\}$,
$\mathcal{S}_D = \{\mathbb{N}_\infty,$ **Bool**, **Entities** $= \wp($**Entity**$)\}$,
$\mathcal{F} = \{F^{\mathrm{Con}}_{s}$: **Connection** $\rightarrow$ **ER Concept**,
F^{Con}_{t} : **Connection** $\rightarrow$ **ER Concept**,
F^{HA}_{s} : **HasAttribute** $\rightarrow$ **ER Concept**,
F^{HA}_{t} : **HasAttribute** $\rightarrow$ **Attribute**,
F^{Comp}_{s} : **Composition** $\rightarrow$ **Attribute**,
F^{Comp}_{t} : **Composition** $\rightarrow$ **Composite Attribute**,
F^{Con}_{minCar} : **Connection** $\rightarrow \mathbb{N}_\infty$,
F^{Con}_{maxCar} : **Connection** $\rightarrow \mathbb{N}_\infty$,
F^{Attr}_{Key} : **Attribute** $\rightarrow$ **Bool**, $F^{Attr}_{Discrim}$: **Attribute** $\rightarrow$ **Bool**,
F^{Re}_{Ident} : **Relationship** $\rightarrow$ **Bool**,
F^{WE}_{owner} : **Weak Entity** $\rightarrow$ **Entities**$\}$,
$\mathcal{R} = \{\in_{En} \subseteq$ **En** $\times \wp($**En**$), \leq_{\mathbb{N}_\infty} \subseteq \mathbb{N}_\infty \times \mathbb{N}_\infty\}$,
$\{0, 1, 2, ..., \infty\}$ of type $\mathbb{N}_\infty$
$\cup \{true(t), false(f)\}$ of type **Bool**.

Furthermore, we accumulated 19 constraints. Again we use the abbreviation ecr for e an entity, c a connection, r a relationship, and c connecting e and r: $ecr := (f^{\mathrm{Con}}_{s}(c) = e \wedge f^{\mathrm{Con}}_{t}(c) = r)$.

$$\forall x \in \mathbf{ERC}\ \exists e \in \mathbf{En}, r \in \mathbf{Re}\ (x = e \vee x = r) \tag{17}$$
$$\nexists e, e' \in \mathbf{En}, c \in \mathbf{Con}\ (f^{Con}_{s}(c) = e \wedge f^{Con}_{t}(c) = e') \tag{18}$$
$$\nexists r, r' \in \mathbf{Re}, c \in \mathbf{Con}\ (f^{Con}_{s}(c) = r \wedge f^{Con}_{t}(c) = r') \tag{19}$$
$$\forall c \in \mathbf{Con}(f^{Con}_{minCar}(c) \neq \infty) \tag{20}$$
$$\forall c \in \mathbf{Con}(F^{Con}_{minCar}(c) \leq_{\mathbb{N}_\infty} F^{Con}_{maxCar}(c)) \tag{21}$$
$$\forall r \in \mathbf{Re}\ \exists c, c' \in \mathbf{Con}$$
$$(f^{\mathrm{Con}}_{s}(c) = r \wedge f^{\mathrm{Con}}_{s}(c') = r \wedge c \neq c') \tag{22}$$

$$\forall a \in \mathbf{Attr}\ \exists h \in \mathbf{HA}\ , c \in \mathbf{Comp}$$
$$((f_t^{\mathrm{HA}}(h) = a \vee f_s^{\mathrm{Comp}}(c) = a) \wedge$$
$$\neg(f_t^{\mathrm{HA}}(h) = a \wedge f_t^{\mathrm{Comp}}(c) = a)) \tag{23}$$
$$\forall a \in \mathbf{Attr}\ h, h' \in \mathbf{HA}\ (f_t^{\mathrm{HA}}(y) = h \wedge f_t^{\mathrm{HA}}(z) = h'$$
$$\Longrightarrow\ h = h') \tag{24}$$
$$\forall a \in \mathbf{Attr}\ c, c' \in \mathbf{Comp}\ (f_s^{\mathrm{Comp}}(c) = a \wedge f_s^{\mathrm{Comp}}(c') = a$$
$$\Longrightarrow\ c = c') \tag{25}$$
$$\forall x \in \mathbf{CA}\ \exists c \in \mathbf{Comp}\ (f_t^{\mathrm{HA}}(c) = x) \tag{26}$$
$$\forall w \in \mathbf{WE}\ \exists r \in \mathbf{Re}\ , c \in \mathbf{Con}\ (wcr \wedge f_{Ident}^{Re}(r) = true \wedge$$
$$f_{minCar}^{Con}(c) = 1 \wedge f_{maxCar}^{Con}(c) = 1) \tag{27}$$
$$\forall r \in \mathbf{Re}\ \exists w \in \mathbf{WE}\ , c \in \mathbf{Con}$$
$$(f_{Ident}^{Re}(r) = t\ \Longrightarrow\ wcr) \tag{28}$$
$$\forall w \in \mathbf{WE}\ , e \in \mathbf{En} \exists r \in \mathbf{Re}\ , c \in \mathbf{Con}$$
$$(e \in_{En} f_{owner}^{WE}(w)\ \Longrightarrow\ wcr \wedge f_{Ident}^{Re}(r) = t) \tag{29}$$
$$\forall a \in \mathbf{Attr}\ \exists w \in \mathbf{WE}\ , h \in \mathbf{HA}\ (f_{Discrim}^{Attr}(a) = t\ \Longrightarrow$$
$$(f_s^{\mathrm{HA}}(h) = w \wedge f_t^{\mathrm{HA}}(h) = a)) \tag{30}$$
$$\forall a \in \mathbf{Attr}\ (\neg(f_{Key}^{Attr}(a) = t \wedge f_{Discrim}^{Attr}(a) = t)) \tag{31}$$
$$\nexists w \in \mathbf{WE}\ , a \in \mathbf{Attr}\ , h \in \mathbf{HA}$$
$$(f_s^{\mathrm{HA}}(h) = w \wedge f_t^{\mathrm{HA}}(h) = a \wedge f_{Key}^{Attr}(a) = t) \tag{32}$$
$$\forall x, y \in \mathbf{ERC}\ c \in \mathbf{Con}\ \exists c' \in \mathbf{Con}$$
$$((f_s^{\mathrm{Con}}(c) = x \wedge f_t^{\mathrm{Con}}(c) = y)\ \Longrightarrow$$
$$(f_s^{\mathrm{Con}}(c') = y \wedge f_t^{\mathrm{Con}}(c') = x)) \tag{33}$$
$$\forall c, c' \in \mathbf{Con}\ ((f_s^{\mathrm{Con}}(c) = f_t^{\mathrm{Con}}(c') \wedge f_t^{\mathrm{Con}}(c) = f_s^{\mathrm{Con}}(c'))$$
$$\Longrightarrow\ (f_{minCar}^{Con}(c) = f_{minCar}^{Con}(c')$$
$$\wedge\ f_{maxCar}^{Con}(c) = f_{maxCar}^{Con}(c'))) \tag{34}$$
$$\forall w \in \mathbf{WE}\ , r \in \mathbf{Re}\ , e \in \mathbf{En}\ , c, c' \in \mathbf{Con}$$
$$((f_{Ident}^{Re}(r) = true \wedge wcr \wedge ec'r \wedge w \neq e)$$
$$\Longrightarrow\ e \in_{En} f_{owner}^{WE}(w)) \tag{35}$$

Similar to the last case study, the ER modeling language is comprehensively captured in a highly concise manner, providing the full information of this data modeling language. This is again proof of the formalism being an extremely precise, concise, and efficient tool for the deployment of modeling methods.

5.2.7 Formalizing graphical models

We formalize the example presented in Fig. 8 and start with specifying the universes of elements of the types **Entity** and **Relationship**. There are three elements of type **Entity**: *Department, Employee*, and *Dependent*, the last one being a weak entity, and two elements of type **Relationship** *has* and *works_in*:

$$\mathcal{U}_{En} = \{Department, Employee, Dependent\},$$
$$\mathcal{U}_{WE} = \{Dependent\}, \mathcal{U}_{Re} = \{has, works\_in\}.$$

Merging these sets results in the universe of elements of the abstract supertype **ERC**:

$$\mathcal{U}_{ERC} = \{Department, Employee, Dependent, has, works\_in\}.$$

We obtain a nesting of universes of elements

$$\mathcal{U}_{Re} \subseteq \mathcal{U}_{ERC} \supseteq \mathcal{U}_{En} \supseteq \mathcal{U}_{WE}.$$

These objects are connected via eight relation elements of type **Con** (to each line the relation in both directions according to constructive constraint (33)):

$$\mathcal{U}_{Con} = \{c_1, c_1', c_2, c_2', c_3, c_3', c_4, c_4'\}$$

with the following source and target elements:

$$f_s^{\mathrm{Con}}(c_1) = Dependent,\ f_t^{\mathrm{Con}}(c_1) = has,$$
$$f_s^{\mathrm{Con}}(c_1') = has,\ f_t^{\mathrm{Con}}(c_1') = Dependent,$$
$$f_s^{\mathrm{Con}}(c_2) = Employee,\ f_t^{\mathrm{Con}}(c_2) = has,$$
$$f_s^{\mathrm{Con}}(c_2') = has,\ f_t^{\mathrm{Con}}(c_2') = Employee,$$
$$f_s^{\mathrm{Con}}(c_3) = Employee,\ f_t^{\mathrm{Con}}(c_3) = works\_in,$$
$$f_s^{\mathrm{Con}}(c_3') = works\_in,\ f_t^{\mathrm{Con}}(c_3') = Employee,$$
$$f_s^{\mathrm{Con}}(c_4) = Department,\ f_t^{\mathrm{Con}}(c_4) = works\_in,$$
$$f_s^{\mathrm{Con}}(c_4') = works\_in,\ f_t^{\mathrm{Con}}(c_4') = Department.$$

One of the two elements of type **Relationship**, *has*, is an identifying one:

$$f_{Ident}^{Re}(works\_in) = false,\ f_{Ident}^{Re}(has) = true.$$

From the model, we read the following cardinality constraints on the connections:

$$f_{minCar}^{Con}(c_1) = f_{minCar}^{Con}(c_1') = 1$$
$$f_{maxCar}^{Con}(c_1) = f_{maxCar}^{Con}(c_1') = 1$$
$$f_{minCar}^{Con}(c_2) = f_{minCar}^{Con}(c_2') = 0$$
$$f_{maxCar}^{Con}(c_2) = f_{maxCar}^{Con}(c_2') = \infty$$
$$f_{minCar}^{Con}(c_3) = f_{minCar}^{Con}(c_3') = 1$$
$$f_{maxCar}^{Con}(c_3) = f_{maxCar}^{Con}(c_3') = 1$$
$$f_{minCar}^{Con}(c_4) = f_{minCar}^{Con}(c_4') = 1$$
$$f_{maxCar}^{Con}(c_4) = f_{maxCar}^{Con}(c_4') = \infty$$

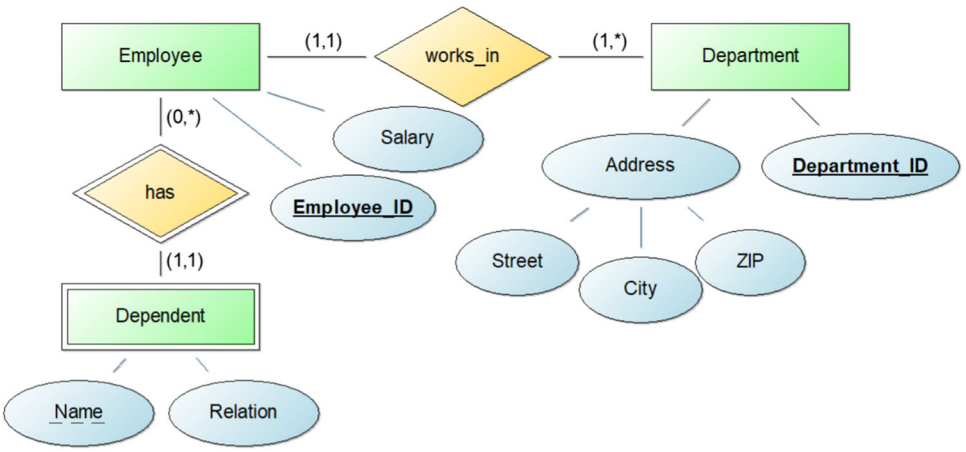


Fig. 8 ER model depicting a data model of employee dependencies

The constructive constraint (35) allows for the automatic derivation of the set of owning entities (in $\wp$(**Entity**)) of the weak entity *Dependent*:

$$f^{WE}_{owner}(Dependent) = \{Employee\}$$

The model contains nine elements of type **Attribute**, one of them is a **Composite Attribute**:

$$\begin{aligned}\mathcal{U}_{Attr} = \{&Employee\_ID(EI), Salary,\\ &Name, Relation, Department\_ID(DI),\\ &Address, Street, City, ZIP\},\\ \mathcal{U}_{CA} = \{&Address\}.\end{aligned}$$

The assignment of these **Attribute** elements to entities and relationships is done via elements in the universe of the **hasAttribute** relation type.

$$\begin{aligned}&\mathcal{U}_{HA} = \{ha_1, ha_2, ha_3, ha_4, ha_5, ha_6\}.\\ &f^{\mathrm{HA}}_s(ha_1) = Dependent,\ f^{\mathrm{HA}}_t(ha_1) = Name,\\ &f^{\mathrm{HA}}_s(ha_2) = Dependent,\ f^{\mathrm{HA}}_t(ha_2) = Relation,\\ &f^{\mathrm{HA}}_s(ha_3) = Employee,\ f^{\mathrm{HA}}_t(ha_3) = EI,\\ &f^{\mathrm{HA}}_s(ha_4) = Employee,\ f^{\mathrm{HA}}_t(ha_4) = Salary,\\ &f^{\mathrm{HA}}_s(ha_5) = Department,\ f^{\mathrm{HA}}_t(ha_5) = Department\_ID,\\ &f^{\mathrm{HA}}_s(ha_6) = Department,\ f^{\mathrm{HA}}_t(ha_6) = Address.\end{aligned}$$

The compound attribute *Address* is composed of three simple **Attribute** elements.

$$\begin{aligned}&\mathcal{U}_{Comp} = \{co_1, co_2, co_3\}.\\ &f^{\mathrm{Comp}}_s(co_1) = Street,\ f^{\mathrm{Comp}}_t(co_1) = Address,\\ &f^{\mathrm{Comp}}_s(co_2) = City,\ f^{\mathrm{Comp}}_t(co_2) = Address,\\ &f^{\mathrm{Comp}}_s(co_3) = ZIP,\ f^{\mathrm{Comp}}_t(co_3) = Address.\end{aligned}$$

Only *EI* and *DI* are primary keys and *Name* is the discriminator for the weak entity *Dependent*. For all other elements in $\mathcal{U}_{Attr}$ the attributes f^{Attr}_{Key} and $f^{Attr}_{Discrim}$ are set to *false*:

$$\begin{aligned}&f^{Attr}_{Key}(Name) = false,\ f^{Attr}_{Discrim}(Name) = true,\\ &f^{Attr}_{Key}(Relation) = false,\ f^{Attr}_{Discrim}(Relation) = false,\\ &f^{Attr}_{Key}(EI) = true,\ f^{Attr}_{Discrim}(EI) = false,\\ &f^{Attr}_{Key}(Salary) = false,\ f^{Attr}_{Discrim}(Salary) = false,\\ &f^{Attr}_{Key}(DI) = true,\ f^{Attr}_{Discrim}(DI) = false,\\ &f^{Attr}_{Key}(Address) = false,\ f^{Attr}_{Discrim}(Address) = false,\\ &f^{Attr}_{Key}(Street) = false,\ f^{Attr}_{Discrim}(Street) = false,\\ &f^{Attr}_{Key}(City) = false,\ f^{Attr}_{Discrim}(City) = false,\\ &f^{Attr}_{Key}(ZIP) = false,\ f^{Attr}_{Discrim}(ZIP) = false.\end{aligned}$$

This concludes the formalization of the model in Fig. 8.

6 Juxtaposition of formalization approaches on different underlying structural theories

In this section, we outline an in-depth juxtaposition of the MetaMorph formalism to three other approaches using different underlying structures. With this comparison, we want to corroborate the choice of logic to be well suited for capturing the characteristics of conceptual modeling languages and also to demonstrate the difference of our approach using formal languages, to be more precise model theory [9], to other implementations based on logic. We proceed with the comparison concept by concept. As candidates for the juxtaposition, we choose FDMM, a representative for a formalism based on set theory, Graph Grammars, a representative based on graph theory, and FORMULA, a representative based on logic besides MetaMorph.

6.1 The approaches

METAMORPH The formalism in the current version was introduced in [16] and presented in Sect. 4 of this paper. It is based on logic and formal languages, to be more precise, on many-sorted model theory.

FDMM The *F*ormalism for *D*escribing ADOxx *M*eta Models and *M*odels [21], FDMM in short, uses set theory to capture the components of modeling languages. It comes with a definition of metamodels and models, although its purpose restricts to providing a formal specification syntax for domain-specific modeling languages implemented on ADOxx.

Graph Grammars To outline the differences between logic-based and graph-based approaches, we consider the employment of graph grammars to define domain-specific modeling languages as introduced in [26, Chap. 10] and neatly formalized in [19]. Thereby, this approach does not introduce a dedicated, closed definition of a modeling language and frequently allows for different implementations to address different requirements, e.g., relations as pairs of vertices or as own elements with source and target assignment, admitting parallel edges.

FORMULA The language called *For*mal *M*odeling *U*sing *L*ogic *A*nalysis [30], FORMULA in short, comprises a formal specification language for DSMLs and a tool offering a proof engine. It builds upon a definition of domain-specific modeling languages based on logic, to be more precise, on universal algebras. The definition of modeling languages and models in this approach does not detail the concrete formalization of object and relation types, attributes, etc., but we found a specification and transformation of MINIMOF sketching the concrete realization of these components (at least one possibility).

6.2 Juxtaposition of concepts

To begin with, we outline the notion of modeling languages and metamodels of the different approaches and describe the formal constructs corresponding to them. Then we will go into detail and examine the formalisms concept by concept.

6.2.1 Modeling languages and metamodels

The concrete usage of the terms modeling language and metamodel show slight differences in the four approaches.

In METAMORPH we define modeling languages as *formal languages* consisting of a *signature* comprising *types/sorts* (in this section we use the term *sort* for a better distinction to *type* as used in conceptual modeling) for object types, relation types, and attribute domains, function and relation symbols, and a set of *constraints*. A model is expressed using a modeling language. A metamodel describes the syntax of a modeling language and is instantiated in a model. The metamodel is itself a model expressed in a metamodeling language. In [16] we introduced the metamodeling language `M2FOL` expressed using METAMORPH.

FDMM targets the formalization of metamodels that are means to capture the syntax. A metamodel in FDMM is a 5-tuple of *modeltypes*, an order relation on the set of *object types*, two functions to map attributes to their *domain* (subset of object types) and *range* (subset of modeltypes, object types, and data types), and a function *card* constraining the number of attribute values. In this formalism, the term *modeling language* is not used.

In graph grammars, the counterpart of a metamodel is an *attributed type graph*, i.e., a graph constituted of (object and relation) types as vertices, source and target assignments, and inheritance (specialization) as edges between these vertices. A model instantiating a metamodel is then a graph labeled with the types of the type graph (i.e., with a structure-preserving mapping to the type graph). Furthermore, type graphs are enriched with constraints via graph constraints and forbidden patterns. This approach furthermore offers a quite different, rather constructive perception of modeling languages, namely as *graph languages* comprising initial valid graphs/models and admissible graph transformations. By applying these transformations, all valid models of the language can be constructed.

In the FORMULA approach, a domain-specific language consists of a *domain* capturing the syntax of the DSML and *interpretations*, i.e., mappings of models of the domain to models in other domains. The domain is a 4-tuple of the two *signatures* Υ, Υ_C (comprising function symbols and arities), an *alphabet* Σ and a set of *constraints*. Thereby, Υ contains function symbols for object and relation types, as well as for attributes. Υ_C is the constraints signature extending Υ and is not relevant in this comparison. The alphabet Σ contains names or identifiers for the modeling elements. Models are then subsets of the universal algebra. That is, all terms are possibly derived by concatenating function symbols and terms. For our juxtaposition, we do not consider the interpretation mappings between models and restrict to the definition of language syntax in the so-called *domain*. Similar to the METAMORPH formalism FORMULA also defines a formal metamodeling language D_{meta}.

6.2.2 Model types

Model types are a means to group object and relation types in different combinations to allow for the creation of models with different views within the same language, e.g., the UML comprises class diagrams, sequence diagrams, etc. FDMM supports this concept and the reuse of object types and attributes in different model types by defining a model type as a set of object types, a set of data types, and a set of

attributes. This leads to a certain degree of complexity, as the same objects or attributes can behave differently in different model types.

The approaches of MetaMorph, Graph Grammars, and FORMULA do not comprise model types and treat them as different languages. Still, the interleaving of different languages (or model types), including tight consistency rules following the idea of a single underlying model, can be addressed with MetaMorph, see [16, 5.2].

6.2.3 Object types

In MetaMorph an object type defines a type (also called a sort) in the signature of the formal language. The set of sorts for object types forms an ordered subset of all sorts. A model, i.e., an $\mathcal{L}$-structure, then contains a universe of elements to each sort (the so-called interpretation of the sort).

There is no set-theoretic concept corresponding to object types in FDMM. Object types are referred to as their own concept. There is no closer description of this concept besides that the set of all object types in a metamodel holds an order relation. This reveals the shortcoming of set theory to not offer the expressive power needed to capture the semantics of types in modeling languages.

In graph grammars, object types T are encoded as vertices in type graphs ($\cong$ metamodel). Objects of type T in a model are vertices in the instance graph "labeled" with the type T. This is realized by a mapping of the instance graph to the type graph, see 6.2.8.

In FORMULA, an object type T can be represented by a unary function symbol f. Applying the function symbol $f(obj)$ to a modeling element obj in the alphabet Σ indicates that the modeling element obj is of type T.

6.2.4 Relation types

In MetaMorph, relation types form a subset of sorts, and each relation type R furthermore defines two function symbols pointing from the sort corresponding to R to the sorts of object types employed as source and target of R.

FDMM considers relation types as special object types with two attributes dedicated to the assignment of elements connected by the relation.

In graph grammars, the range of possible realizations extends from (ordered or unordered) edges as pairs of vertices for binary, simple (non-parallel) relations to autonomous elements connected by edges to a source and a target vertex for multigraphs (admitting parallel edges). In the type graph, they are represented as vertices with two edges to the source- and target type. Graph grammars furthermore offer the prospect of n-ary relations realized as hypergraphs.

In the FORMULA approach, relation types can be encoded as ternary function symbols in the signature Υ. The arguments of the function are the name of the relation (an element in the alphabet Σ), the source and the target (any terms in the term algebra), i.e., relations are considered binary.

In summary, all four approaches admit multiple relations of the same type between the same two elements.

6.2.5 Specialization

Specialization is also termed *inheritance* in some approaches.

MetaMorph introduces specialization on object types but not on relation types. Specialization imposes an order relation on the sorts for object types indicating a containment relation between the corresponding universes of elements in an instance model of the language.

FDMM also includes an order relation on the set of "object types" indicating the inheritance. As relation types are treated as a subgroup of object types, they can also inherit from other types.

In graph grammars, a type graph can be equipped with inheritance relations between types. To realize the proper behavior of this relation, the procedure of an (abstract or concrete) closure of the type graph is described [19]. A model is then an attributed instance graph typed over the closure of the original type graph. We did not find any restriction of specialization for object types.

Although the concept of specialization appears in the metamodeling language D_{meta} of the FORMULA approach, we could not detect its realization in the signature of a domain.

6.2.6 Attributes and value domains

MetaMorph defines attributes as mappings of the attributed element—objects or relations—to the value domain of the attribute. This can be a constant value, another modeling element—object or relation—multiple values at once, or even a complex construct comprising several of the possibilities just mentioned. This is done using so-called product types, i.e., cartesian products of simple types. For constant value types and values, the signature comprises a subset of sorts and constants dedicated to a concrete sort.

In FDMM, a model type comprises sets of "attributes" and offers "data types". Similar to object types, there is no corresponding concept from set theory for these terms. Attributes can be mapped to data types as well as object types, and model types. In contrast to MetaMorph, the FDMM approach introduces a standalone element in the formalism representing an attribute instead of a mapping to the attribute value. An attribute can therefore be assigned to many object types or model types, but its domain and range are fixed globally. It is named as the task of the engineer to care for a further structure of a "data set" and enrich it in some way with concrete values. The cardinality function of a metamodel

assigns a minimal and maximal amount of possible values to an attribute. Attributes are therefore used as multi-valued.

In graph grammars, there are again two ways of specifying attributes. One way is to add to the graph a function per attribute pointing to the attribute's value domain deviating from graph theory. Another possibility is to create possible attribute values as own data vertices and to introduce an "attributing edge" between an object or relation and the data for the concrete assignment. This somehow blows up the graph with elements usually considered subordinate for the model. By adding more than one attributing edge per attribute, a multi-value attribute can be realized. All in all, both ways show that graph theory does not offer a straightforward way to introduce attributes.

In the FORMULA approach, attributes can be specified as binary function symbols relating a modeling element with a concrete attribute value. The value domains of attributes are limited to bool, string, and enum, whereas for the enumeration, the possible values have to be specified. This hints at the perception of attributes being single-valued.

6.2.7 Constraint language

In principle, any type of logic can be used to encode constraints for possible interpretations of a formal language signature. For METAMORPH, we choose first-order logic due to its expressivity and the familiarity of most engineers with FOL. Given a signature and a set of constraints, the core subject of model theory, basis of METAMORPH, is to find interpretations of the signature that fulfill these constraints, i.e., valid models.

The FDMM approach does not comprise a method to capture constraints on a metamodel.

Graph Grammars offer two methods to encode constraints. One of them is to graphically specify forbidden patterns (negative constraints). The second one specifies conditional constraints using logical quantifiers on graph snippets to ensure the required constellations. With this type of constraint, one can also ensure the existence of a concrete pattern (positive constraints).

Similar to METAMORPH, the logic-based approach FORMULA canonically encompasses a constraint language. To meet the needs of the theorem prover, Horn Logic was chosen.

6.2.8 Instantiation

In METAMORPH we make use of the canonical correspondence of instantiation between the metamodel and the model and the relation of a formal language $\mathcal{L}$ with signature Σ and an $\mathcal{L}$-structure interpreting the signature. Also, specialization as containment of universes is canonically realizable with order-sorted model theory and containment of universes.

FDMM defines instantiation between the metamodel and an instance model as a mapping of model types to sets of model instances and functions, taking object types to collections of objects, as well as data types to the data objects. In terms of set theory, this is a function like any other, therefore not inherently offering the semantics of the instantiation dependency between type and instance.

In graph grammars, an instance graph conforms to a type graph if there is a mapping of the instance graph to the type graph that preserves the graph structure. In contrast to FDMM and the concept of instantiation, this function points from instance- to meta-level and brings along a constraint for the proper behavior of the mapping. Still, the constructs of a type graph, instance graph, and the function from the latter to the former preserving the graph structure are not congruent to the semantics of metamodel instantiation. Structure-preserving mappings between graphs are too general for instantiation.

In the FORMULA approach, we have to build the universal algebra of the signature Υ and the alphabet Σ. The universal algebra is the set of all possible terms (elements in the alphabet and function symbols applied to the alphabet and other terms). The set of possible models is then a subset of the power set of the universal algebra, namely those subsets complying with the well-formedness rules. Notice that this approach differs from model theory used in METAMORPH as the signature is not interpreted. This means that there are no concrete objects in a language structure, and the terms are uninterpreted, i.e., $f(x)$ is a term standing for itself and is not mapped to another element in the algebra. The possible elements in a model are already "named" in the alphabet Σ.

6.2.9 Model

A model in the METAMORPH formalism is an $\mathcal{L}$-structure conforming to the constraints of the language, i.e., a structure comprising a universe of elements for each sort and interpretations of the function- and relation symbols in Σ on these universes, such that all constraints of the language are fulfilled.

In FDMM, a model is a 5-tuple consisting of the three mappings described above (6.2.8) and triples of an object element, an attribute element, and the concrete attribute value, as well as an assignment of model instances to a set of these instances. This complexity arises as object types and attribute types can be reused in different model types.

In graph grammars, models of a language are so-called typed, attributed instance graphs with a structure-preserving mapping to the type graph that fulfill all the language constraints.

The FORMULA formalism defines a model as a subset of the universal algebra, i.e., a subset of all possible terms. These terms must fulfill the language constraints.

6.3 Summary and résumé

The results of the comparison are collated in Table 1. Summarized, this comparison confirms the choice of formal languages and model theory.

Set theory lacks correspondents for crucial concepts in conceptual modeling like object types. As the single goal of FDMM is to offer a specification language without a further investigation of the formal kernel of conceptual modeling this weakness of its structural basis has no consequences. Nevertheless, the absence of a constraint language makes an exhaustive and complete apprehension of a modeling language impossible. Also, the omnipresent definition of mappings between sets of vague concepts not matching the semantics of conceptual modeling notion questions the suitability of set theory for providing accurate concepts.

Graph grammars are well suited to represent the graphical box-and-line-like appearance of models. In addition to that, this approach adds a lot of structure to vertices and edges of graphs to include also those concepts of conceptual modeling going beyond the box-and-line structure. In the case of attributes and inheritance, this is not a straightforward process and leads to cumbersome additions making use of other structural theories like algebra and logic. Therefore, we doubt that graphs are best suited for the task of formalizing modeling languages to their full extent.

The logic-based approach FORMULA serves well the purpose of formalizing the complete syntax of a modeling language in a straight manner, but it also shows some impracticalities. The (one-dimensional) concatenation of function symbols and terms does not intuitively reflect the two-dimensional nature of models. Furthermore, it lacks in the version at hand a suitable method to implement inheritance and to define attribute value domains in an adequately free manner. These value domains can become quite complex, e.g., the methods of a class in UML class diagrams comprise input parameters and return values.

MetaMorph, the formalism presented in this paper, covers the most prominent concepts constituting conceptual modeling languages. Logic, formal languages, and model theory offer a straightforward method to canonically capture the semantics of types, attributes, and instantiation and also provides meaningful methods to address advanced concepts and practices like power types and language interleaving, see [16]. Our approach does not cover all aspects of modeling languages yet, e.g., specialization on relation types, model types, or derived types. Still, the results achieved so far are encouraging that the definition of formal modeling languages at hand thoroughly serves the purpose and holds the potential for further advancement according to the needs of conceptual modeling research.

7 Empirical assessment

To evaluate the suitability of MetaMorph also from the practical perspective and to prevent alienation from practitioners' intuition and needs, it is important to involve language engineers for feedback on the formalism design. As mentioned before a formalism is only useful for practicing language engineering if it adequately reflects the characteristics of modeling languages in a domain-independent way and if its application is intuitive and reasonable in terms of effort. Only then can it support practitioners in their work and contribute to the maturation of the scientific field.

To achieve this involvement of practitioners, we conducted an empirical assessment with newly trained language engineers at university. The formalism, as introduced in Sect. 4, was provided to a group of students in the course "Metamodeling" at the University of Vienna. Students were asked to formally define their own domain-specific language developed throughout the course and assess the capabilities provided by the formalism.

In the following subsections, the underlying research questions for this evaluation, the methodology used, and the results are discussed.

7.1 Research questions for empirical assessment

The goal of the empirical research is to assess and evaluate the use of MetaMorph together with language engineers to get feedback on the adequacy and usability of the formalism and further refine the provided constructs. Feedback received influences the development in an iterative manner.

The research questions established for the empirical assessment are introduced below:

1. Does the formalism adequately reflect conceptual modeling languages?
   Objective Completeness. Evaluate whether the formalism provides all required concepts to define a language and model from a practitioner's perspective.
   Subquestions Is the formalism complete on metamodel level? Is the formalism complete on model level? Does MetaMorph comprise irrelevant concepts?
2. Does the application of the formalism appear intuitive to language engineers?
   Objective Intuitiveness. Evaluate whether the concepts are fairly intuitive for language engineers, fit their conception of modeling languages, and can be applied in a practical project.
3. Does the formalization of a modeling language have influence on the modeling language design?
   Objective Design. Evaluate whether the formalization requirement changes design decisions.

Table 1 Summary of the comparison of formalization approaches (o := not available)

	MetaMorph	FDMM	Graph grammars	FORMULA
Structural theory	Logic, Many-sorted model theory	Set Theory	Graph Theory	Logic, Universal Algebras
Modeling Language	Formal language $\mathcal{L}$ with signature Σ and a set of constraints	o	Graph language	Signature Υ with alphabet Σ and a set of constraints
Metamodel	A model expressed in the formal metamodeling language M2FOL [16]	A tuple of: modeltypes, an order relation on object types, and domain-, range-, and card mappings for attributes	An attributed type graph with inheritance	A model expressed in the formal metamodeling language D_{meta}
Object Types	Types/Sorts	"Object types", no corresponding concept in set theory	Vertices in a type graph	Unary function symbols taking as input an element from the alphabet
Specialization/ Inheritance on Object Types	Order relation on sorts of object types	Order relation on the set of "object types"	Inheritance relation in type graph and "closure" of type graph	?
Relation Types	Sorts with two function symbols assigning source and target	Special object types with additional attributes for source and target assignment	Edges: elements with source and target mappings	Ternary function symbol taking as input an element from the alphabet, the source and the target
	Binary	Binary	Binary; n-ary potentially realizable with hypergraphs	Binary
Specialization/ Inheritance on Relation Types	o	Via inheritance on object types	?	?
Attributes	Functions from an object or relation type to the value domain	"Attributes", no corresponding concept in set theory	Data vertices with attribute edges connected to attributed object or relation	Binary function symbol taking as input a modeling element and the concrete attribute value
	Single- and multi-valued	Single- and multi-valued	Single- and multi-valued	Single-valued
Value Domains	Sorts for value domains and constants assigned to these sorts	"Data types", no corresponding concept in set theory, no general outlined procedure on how to specify concrete data values	Data values added to a graph as own vertices	Fixed: bool, string, and enum
Constraint Language	First-Order Logic	o	Forbidden patterns and logical expressions on graph snippets	Horn Logic
Instantiation	Interpretation of the signature Σ of the language $\mathcal{L}$	Functions of model types to sets of model instances and object types to collections of objects	Mapping of the instance graph to the type graph that preserves the graph structure	Elements in the power set of the universal algebra over the signature and alphabet are possible model instances of a DSML
Model	$\mathcal{L}$-structure conforming to the constraints	Tupel of mappings between elements of the metamodel(see above) to a set of "model instances", sets of objects, sets of data objects	Typed attributed instance graphs conforming to the constraints	Subset of the universal algebra conforming to the constraints

4. How do language engineers experience the complexity of learning the formalism? Do they experience a difference in the complexity to formalize the different aspects of a language or a model?
   Objective Complexity. Evaluate the learning effort to gain expertise in using MetaMorph and the difficulty to formalize different concepts.
5. Do the skills of the language engineer influence the experienced difficulty of using MetaMorph?
   Objective Skill. Impact of skills/competencies of the user on the application of the formalism.
6. How much time does it take a language engineer to formalize small to medium-sized languages?
   Objective Usability. Evaluate the time needed for formalization as well as influence factors on the duration.
   Subquestions Does the language size influence the time needed for formalization? Does the experienced difficulty influence the time needed for formalization?
7. How can language engineers be assisted to successfully apply MetaMorph?
   Objective Support Requirements. Evaluate the need for support. E.g., does an interactive introduction have a positive influence on the use of MetaMorph?

These research questions address different qualities of MetaMorph:

RQ1 and RQ2 evaluate the *adequacy* of the construction of the formalism from a practitioner's view. This includes the completeness of concepts to define a modeling language in real-world cases, as well as the appropriate congruence and conformity to the characteristics of conceptual modeling languages becoming apparent in an intuitive application. An adequate formalism must not compel the language engineer to an intricate line of thinking but encourage a straightforward formalization process.

Usability is addressed by RQ4 and RQ6 by evaluating the experienced complexity in applying the formalism as well as the effort needed to formalize a complete language. Also, the intuitive handling of the formalism (RQ2) contributes to convenient usability. This allows for conclusions on the efficiency and satisfaction of using MetaMorph for formalization.

RQ5 and RQ7 furthermore aim to assess the required background and minimal pre-knowledge for a successful application of the formalism and to determine suitable assistance in the form of materials and training options to optimally support the user. From this, we can learn how to improve usability for future users of MetaMorph.

7.2 Methodology and evaluation setup

As a methodological baseline, the assessment design in Wohlin et al. [60] was selected using a questionnaire to gain insights into the participants' experience with the formalism.

7.2.1 Choice of cohort

For the conduction of the assessment we decided to work with newly trained language engineers from the University of Vienna. The benefits of acquiring participants for the evaluation directly at a university course teaching metamodeling are manifold: First, these courses are attended by a decent number of potential participants, and the students are directly accessible for personal interaction. This simplifies the organization and the support of participants. An acquisition of participants by calls via mailing lists would complicate the administration of an introduction session and usually results in a low answer rate.

Second, students coming directly from a training in language engineering have similar preconditions and provide us with directly comparable feedback. They are on the same level of knowledge about language engineering and are unbiased regarding a domain and existing formalisms already used. This eliminates the necessity to profoundly consider their background of research and possible influences on the feedback.

Third, the languages designed in the university course are approximately same sized. Therefore we can exclude the language dimension as a reason for possible differences in effort and investigate alternative reasons.

We asked the participants to formalize self-designed methods. This prevents unnecessary difficulties in the formalization process arising from the need to become acquainted with a new language. As students formalized their own language, they are well aware of the concepts, semantics, and constraints, as well as the design process. Therefore, they also have the ability to recognize possible weaknesses and potential improvements of the design revealed in the formalization process.

Of course, these novices don't have the same expertise with language engineering as scientists and practitioners who have spent significant time with the development and application of modeling languages. The assessment conducted does not claim to be complete, and we are striving for the feedback of experienced engineers. In our opinion, such feedback can be collected more efficiently in direct collaborations than in unilateral questionnaires. To obtain this, we plan to con-

duct various projects on formalization in the setting of the OMiLAB [47] together with experts who bring their own modeling methods. Currently planned are collaborations on the Scene2Model method [43] and the Dig4Biz method [54]. Nevertheless, the perspective of a novice is also important since the formalism should not be comprehensible only to advanced engineers.

7.2.2 Setup and execution

The course "Metamodeling" within the master's program of "Business Informatics" at the University of Vienna in the summer term of 2021 was selected as the environment to perform the above study. Participation was optional, and 27 students participated in the assessment. As an incentive, 7 points of the course were offered to participants who completed the tasks outlined individually and seriously.

Students were informed that the evaluation is conducted optional and anonymized to prevent feedback motivated by social desirability. Correctness was no requirement for receiving full points, and the students were aware that their experience with the formalism was relevant to the study, not the correctness or completeness of the outcome. Therefore, students had no incentive to collaborate or copy from colleagues.

The written description of a modeling language developed in advance within a group project was utilized as input for the assessment. This implies that all participants already had (a) learned about the foundation of metamodeling and language design, (b) applied it practically on a self-selected case, and (c) did not implement this case on a metamodeling platform yet.

From a methodological viewpoint, the assessment was scheduled in four phases:

1. *Introduction to* METAMORPH
   As a warm-up, the participants received an introduction to METAMORPH, starting with a recap on sorted logic and a detailed explanation of the formalization on language and model level. This theoretical introduction was accompanied with a practical, interactive demonstration, applying METAMORPH to Petri Nets as discussed in [15] and to PROVIS as presented in Sect. 5.1. Additionally, as a reading assignment, the students received background literature [18] acting as a walk-through for their own formalization process.
   The introduction was offered to all participants to give them the background required for the rating of intuitiveness and completeness of the formalism. As a consequence, no control group for a comprehensive comparison of results is available. Therefore for the comparison required in research question 7, we used those participants that could not join the introduction as a proxy for a control group.
2. *Formalization Case*
   During this phase, the participants were asked to formalize the modeling language that they had designed as part of the course assignments. The design had been performed in groups (4–5 team members per group). The formalization assignment was an individual task performed by every participant. As a support instrument, the participants received a prepared document template with the required logical symbols for the subsets of the signature to support the task. The participants were given two weeks to submit the results.
3. *Evaluation and Feedback*
   As a concluding phase, participants were asked to fill out an online questionnaire about their experience during the formalization phase. The participants had two weeks to reflect and provide their feedback.
4. *Analysis of Feedback*
   The feedback given by students was anonymized and filtered for deficient inputs. The data evaluation was conducted using the statistical programming language R.

7.2.3 Participant demographic

Twenty-seven students volunteered to participate in the assessment. Thirteen had completed their bachelor's degree at the faculty of computer science of the University of Vienna, and the rest joined the master's program from other universities. Several of them had completed their studies in business-related programs and degrees at the University of Vienna or somewhere else.

As an initial step, the participants were asked to perform a self-assessment of their pre-knowledge in modeling, metamodeling, and predicate logic as a means to contextualize the results of the evaluation phase. Figure 9 shows the results of the self-assessment graphically.

On a scale from 1 (best) to 6 (worst), they rated their skills in modeling and predicate logic as slightly positive (Mean 3.04 and 3.04 respectively) and their skills in metamodeling as slightly negative (Mean 3.68).

For the evaluation, 2 participants had to be excluded due to missing or incomplete formalization results (Phase 2 of the empirical study); 4 students did not attend the introduction session (Phase 1) and worked solely with the provided documents.

7.2.4 Limitations of empirical assessment

For the empirical validation of the formalism, we identified beforehand the following limitations.

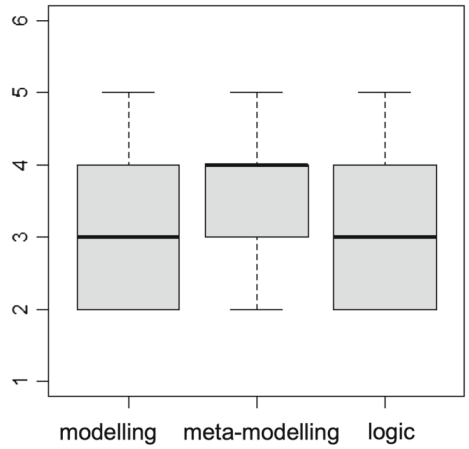


Fig. 9 Self-assessment: Skills in modeling, metamodeling, and logic, 1=best, 6=worst

- Small number of participants: as an assessment, only a limited number of participants was selected to participate. Extension to a larger group would elevate the data set and potentially provide further insights.
- Participant skill level: the evaluation was performed with students and not metamodeling experts.
- Self-assessment of skills within a university course is potentially not objective.
- Online setting: Doing the formalization via an online system was rather laborious because of the necessity to write logical symbols in a digital document (which was also critiqued by students). This was necessary due to the online format of the course.
- Social bias: Although the evaluation of the feedback was anonymized and optional, still, a subliminal bias towards the social desirability of favorable answers might emerge for some students.

As such, the assessment organized in the course metamodeling provides initial insights to continuously improve the concepts contained, identify issues early in the design process, and inform future releases.

7.3 Results

The results of the evaluation phase are discussed in the following, based on the results submitted by participants. The research questions (RQ) are discussed and recommendations from the evaluation are deduced for each question.

RQ 1 Does the formalism adequately reflect conceptual modeling languages?

Completeness on metamodel level: Does the formalism provide all the concepts needed to formalize a modeling language?
Answers collected: 4% (1) No, 20% (5) central concepts and almost all others are covered, 76% (19) Yes
Missing concepts mentioned by "central concepts" voters are:

- Concepts for areas or regions with spatial extension (2x)
- Maps or concepts for spatial modeling (1x)
- Functionality (1x)

Disclaimer: the "no" vote results from a misunderstanding of the participant as he/she did not formalize all concepts of the language designed.
Conclusions: The coverage of concepts in the formalism is satisfactory. Mentioned missing concepts are related to spatial modeling, which concerns the general applicability of the formalism as a domain-specific concept.
Completeness on model level: Does the formalism provide all the concepts needed to formalize a model?
Answers collected: 96% (24) Yes, 4% (1) No
Missing concepts mentioned by "No" voters are:

- Notation

Conclusions: The coverage of concepts in the formalism is satisfactory. Notation, and more specifically graphical notation, is considered a secondary artifact for formalization.
Does MetaMorph *comprise irrelevant concepts?* (80% (20) No, 20% (5) Yes)
Mentioned irrelevant concepts of languages:

- Multi-value attributes (4x)
- Additional symbols (1x)
- Some specific concepts of students' languages

Conclusions: Several participants see multi-value attributes as superfluous and argue that the same result can be achieved with single-value attributes, even though they are indeed necessary for some languages, see, e.g., the example of ProVis presented in the paper at hand.

RQ 2 Does the application of the formalism appear intuitive to language engineers?

The participants provided feedback on whether the application is intuitive, therefore assessing the usability of it in concrete projects. Figures 10 and 11 provide a graphical representation of the results (1=best, 6=worst, Mean: 2.84, Median:3, Standard Deviation: 1.179)

80% (20) of participants rate the intuitiveness as rather positive to very positive (range 1–3). No one rated absolutely negative (6).

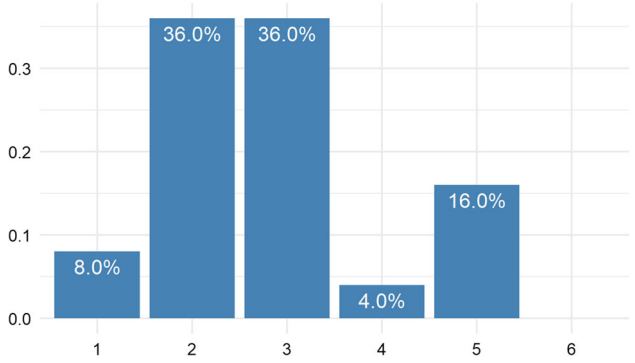
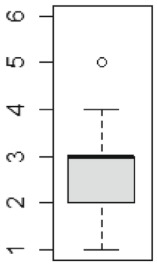


Fig. 10 Ratings of intuitiveness, 1=best, 6=worst

Fig. 11 Ratings of intuitiveness, boxplot, 1=best, 6=worst



Conclusions: We see this as positive feedback that METAMORPH can be applied practically.

RQ 3 Does the formalization of a modeling language have influence on the modeling language design?

36% (9) of students mentioned that they had to explicate further details of their language to be able to completely formalize it, whereas 64% (16) already specified all details before and had a fully determined language as input.

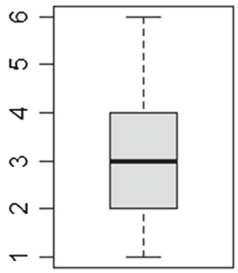
Details that had to be (re-)designed during the formalization include:

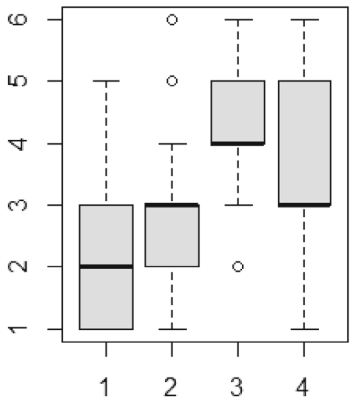
- Constraints (4x)
- Value domains (2x)
- Attributes (5x)
- Relation types (2x)

44% (11) of the participants reworked their language design due to new considerations coming up in the formalization process, whereas 56% (14) did not see the need for design changes in their languages. Participants who re-designed mentioned that the formalization triggered inspiration to reflect the current design and rework it (e.g., identified shortcomings in the design, superfluous attributes).

Conclusions: The process of formalization can trigger design re-considerations and reveal shortcomings in the initial design before it comes to the implementation phase.

Fig. 12 Experienced complexity to learn METAMORPH, 1=best, 6=worst





	1: object- and relation types	2: attributes and value domains	3: constraints	4: models
Mean	2.16	2.84	4.16	3.56
Std. Dev.	1.180	1.344	1.313	1.474
Median	2	3	4	3

Fig. 13 Experienced complexity by concept type, 1=best, 6=worst

RQ 4 How do language engineers experience the complexity to learn the formalism? Do they experience a difference in the complexity to formalize the different aspects of a language or a model?

The experience of learning the formalism and its inherent complexity in understanding and applying the concepts is considered slightly positive (Min 1, Max 6, Mean: 3.24, Median:3, Standard Deviation: 1.268). Figure 12 provides a graphical representation of this outcome.

Nevertheless, there are extensive differences in the difficulty to formalize the different concepts, see Fig. 13.

The concepts easiest to formalize are object- and relation types; the hardest is the formulation of constraints. In addition, models are not easy to formalize for participants.

Conclusions: The complexity of learning the formalism is rated slightly positive. The complexity to apply METAMORPH to constraints and models is rated rather high to high. This evaluation is interpreted as a need to further extend

introductions and tutorials and potentially realize support tools/methods for METAMORPH.

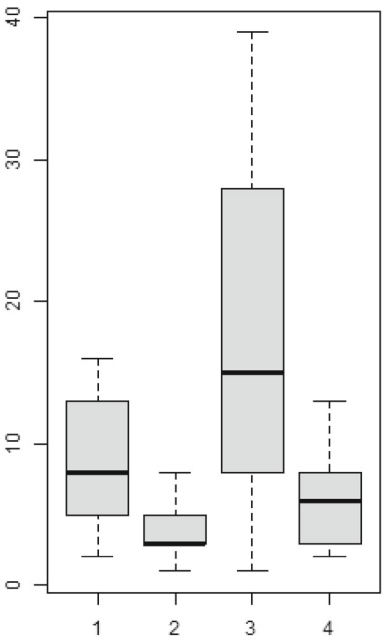
RQ 5 Do the skills of the language engineer influence the experienced difficulty of using METAMORPH?

The aspect of skill and competencies is addressed by this research question. A high, significant correlation (0.42, p=0.036) between the self-assessed skills in metamodeling and the experienced complexity to learn METAMORPH could be identified. Figure 14 shows the results graphically.

Regarding the experienced complexities, the following observations apply:

- A significant correlation (0.48, p=0.015) between the self-assessed skills in metamodeling and the experienced difficulty to formalize models can be found.
- There is a significant correlation (0.41, p=0.04) between the self-assessed skills in predicate logic and the experienced difficulty to formalize models.
- There are no other considerable significant correlations between skills and experienced difficulties.

Table 2 Numbers of object types, relation types, attributes, constraints of participants' languages



Number of	1: object types	2: relation types	3: attributes	4: constraints
Min	2	1	1	2
Max	16	8	70	13
Mean	8.4	4.12	18.56	6.2
Median	8	3	15	6

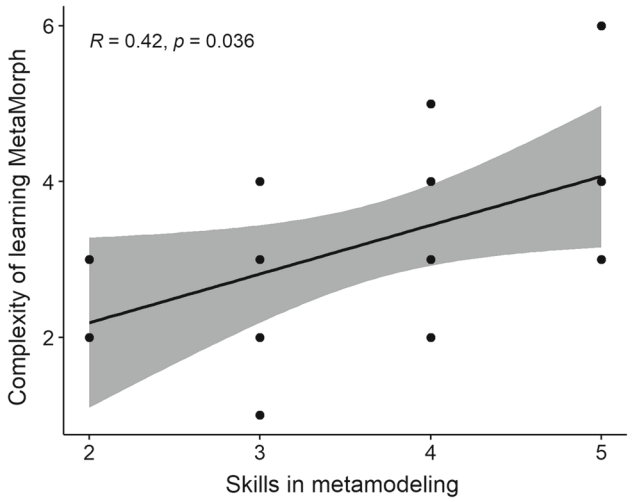
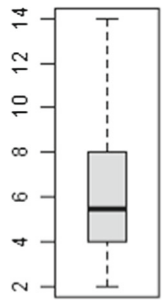


Fig. 14 Correlation: complexity of learning METAMORPH and skill self-assessment in metamodeling

Fig. 15 Assessment of time effort related to formalization



- Interestingly, there is also no significant correlation between skills in predicate logic and the difficulty to formalize constraints (0.30, p= 0.1396).

Conclusions: A better basis in metamodeling leads to an easier acquisition of the formalism. Good skills in predicate logic can help in some aspects of the formalism founded in logic (models) but not in others (constraints).

RQ 6 How long do language engineers need to formalize small to medium-sized languages?

The participants were asked for the duration of the formalization process. 50% of the participants spent between 4 to 8 hours with the formalization (Min 2, Max 14, Mean: 5.94, Median: 5.5, Standard Deviation: 2.844), see Fig. 15. The objective of this research question relates to the effort for formalization and influence factors thereon.

Influence of language size: Does the language size have an influence on the time needed for formalization?
In Table 2 the language sizes split into concepts are shown. The assessment data shows that in the responses by participants, there is no correlation between time and the number of concepts of the language(-0.095, p=0.65), which implies that the formalization effort is not directly related to the number of constructs designed.

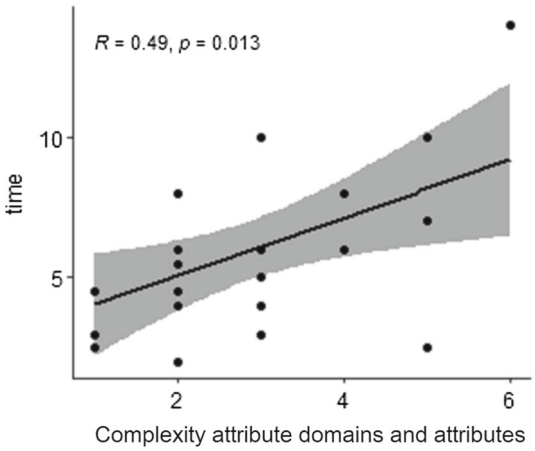


Fig. 16 Correlation: Complexity of attributes and time effort

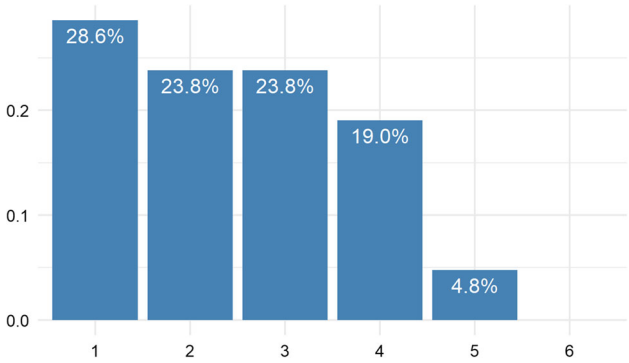


Fig. 17 Introduction Assessment, 1 = very helpful, 6 = not required

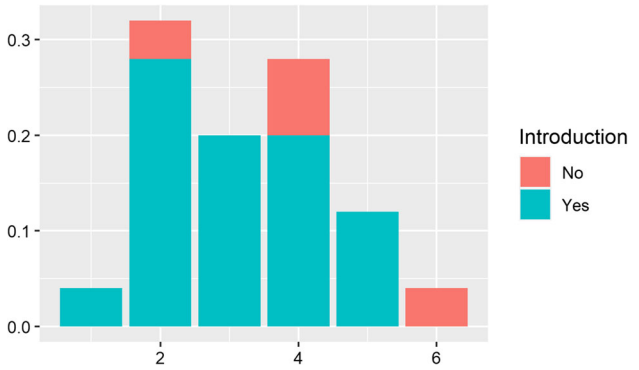


Fig. 18 Complexity to learn METAMORPH split into attendees and non-attendees of introduction, 1=best, 6=worst

Table 3 Mean values of intuitiveness and experienced complexity of different concepts compared between attendees and non-attendees of the introduction

Mean	Attendees	Non attendees
Complexity to learn METAMORPH	3.10	4
Intuitiveness	2.86	2.75
Complexity of object- and relation types	1.95	3.25
Complexity of attribute domains and attributes	2.76	3.25
Complexity of constraints	4.05	4.75
Complexity of models	3.29	5

Influence of complexity: Does the experienced difficulty have an influence on the time needed for formalization? Drilling down into the data of the evaluation, we can recognize that the experienced difficulty to formalize value domains and attributes influences the time. Figure 16 shows that the experienced complexity has a significant correlation (0.49, p=0.013) towards time effort spent, whereas no other concept has a noteworthy influence.
Conclusions: The aptitude of the engineer, i.e., a low experienced complexity, has a bigger influence on the time needed for formalization than the actual language size.

RQ 7 How can language engineers be assisted to successfully apply METAMORPH? Does an interactive introduction have a positive influence on the use of METAMORPH?

The essence of this research question is to assess whether a guided walk-through and tooling are required to understand and apply the formalism.

Those who attended rated the helpfulness of the introduction according to Fig. 17 (1 = very helpful, 6 = not required, Mean: 2.48, Median 2).

4 of the 25 participants did not attend the introductory session on METAMORPH and used solely the definitions and examples from [16] and [18] to learn the formalism independently. Therefore the assessment results of this group are contrasted with those participating in Phase 1 of the assessment targeting the question of whether the introduction had an influence on the experienced complexity and intuitiveness. The results show that participants attending the introductory session found it easier to learn the formalism (see Fig. 18), but there is nearly no difference in the rating of intuitiveness (see Table 3). Several concepts we experienced less complex after attending the introduction. These are object- and relation types, and models whereas we consider an impact for those elements where the difference in the mean is larger than 1.

The above result is substantiated by the responses to the question of what means would elevate the application and use of the formalism. The result shows the need for extended background knowledge in logic (56%, 14), better knowledge in metamodeling (32%, 8), and the need for additional practice and examples (84%, 21).
Conclusions: To support language engineers in applying METAMORPH sophisticated training material, video tutorials, and interactive training are highly recommended.

7.4 Overall conclusion

In summary, we draw the following conclusions for the qualities of MetaMorph:

Regarding *adequacy*, the results indicate that the formalism meets the needs of practitioners. Those concepts mentioned as irrelevant, i.e., multi-valued attributes and additional concepts, appear to be necessary as soon as we intend to formalize sophisticated languages like ProVis or UML. The estimation as irrelevant might stem from the inexperience of novices. This is also indicated by RQ5, showing that a better knowledge of metamodeling results in an easier acquisition of the formalism. The intuitiveness comes off well, even better than the difficulty of learning the formalism and the complexity to formalize the different concepts. Altogether we are encouraged that the foundation and design of MetaMorph are suitable and conformant to the structure of modeling languages also from the practitioner's view. The intuitiveness is independent of an interactive introduction, which is an indicator to be thoroughly fitting to participants' intuition.

According *usability* we recognize that the complexity to get acquainted with MetaMorph and the complexity to formalize different concepts is rated slightly positive and less to rather hard. Thereby, the experienced complexity has a bigger influence on the time needed for formalization than the actual language size. This is a strong indicator of the need for more elaborate, supportive material and training options so that users are helped to overcome the initial obstacle of complexity. This could probably lead to a solid improvement of time effort needed for formalization. This is also backed by the feedback on the introductory lecture, that 76% rated very helpful to rather helpful, and that those not participating in the introduction rated the difficulty to learn MetaMorph higher than those participating.

The case study, as well as empirical work, has impacted the development of the formalism, discussed in the following section as lessons learned during this research iteration of MetaMorph.

8 Lessons learned

From the extensive case studies presented in this paper and from previous formalization projects, as well as from the empirical assessment, we can record four recurring observations.

The most salient observation is that the expressiveness and complexity of a language are captured in the concepts commonly considered subsidiary to object types, namely relation types, attributes, and constraints, rather than in the object types themselves. Although object types are the most prominent element in a metamodel, the relation types and the "hidden" constructs of attributes and constraints do a better job of capturing processable semantics of models. This is reflected in the higher effort of formalizing these concepts according to Definition 1 and is underpinned by a number of case studies on expressive modeling languages with few object types but intricate constraints, e.g. Petri Nets [15], ER diagrams and ProVis, or complex multi-value attributes, e.g. ProVis and UML class diagrams [16]. Also, the results from the empirical evaluation, research question 4 back this claim.

Furthermore, we observe that constraints can be deployed for two opposing purposes. The first one is a restrictive one, i.e., limiting the set of valid model constructs. The second one is a constructive one, i.e., enforcing the existence of additional modeling elements or determining attribute values. Examples of the first type are cardinality constraints or the restrictions for tree-like structures without circles. Examples for the second type are rules for attribute values like the numerous dependencies of frequencies in unit squares or the execution time of whole processes depending on the single tasks. In this sense, the constructive usage can be seen as a generating functionality of a language.

The third observation concerns the interrelationship of the graphical and the formalized representation of a model. Of course, the spatial appearance is a crucial point for human comprehension in conceptual modeling but is mostly omitted in the formalized representation of a model. Nevertheless, spatial arrangement and formalization can have a pertinent mutual impact. In the case study on ProVis, the graphical manifestation of a unit square is highly dependent on the attributes of conditional probability determined by the numerous constructive constraints. In contrast to that, in UML sequence diagrams, the vertical ordering of messages from a lifeline determines the anteriority and posteriority of messages, which definitely has to be captured in a formalized model.

Evaluating the empirical assessment, we made our last observation concerning the impact of the formalization process on design decisions. 44% of participants mentioned that they reworked the language design due to new considerations. The reasons mentioned were insufficient preciseness in attribute definition or a need for further constraints. This firstly underpins the importance of formalization in the language development process and secondly complies with the first observation as it again reveals that it is the "subsidiary" concepts that bear the expressive power of a language and therefore require more consideration than the object types.

9 Conclusion, limitations, and outlook

The paper at hand gives a pervasive description of our formalism MetaMorph proposed in [15] and [16] and illustrates its adequacy and usability by applying it to two extensive case

studies, by comparing the approach to formalisms with similar goals, and by collecting feedback from student language engineers in an empirical assessment.

For the case studies, we chose the modeling method PROVIS – *Probability visualized* – from the domain of stochastic education and the well-known Entity-Relationship language. The distinguishing feature of both languages is the low amount of object and relation types but high interconnectivity of concepts and their attributes. This makes them highly interesting specimens for formalization. Conducting the case studies, the formalization of all language concepts is shown step by step.

To underpin the appropriateness of formalism design, we gave a detailed juxtaposition of METAMORPH with three other formalization approaches based on different foundational theories, i.e., set theory, graph theory, and logic. In this comparison, we determine the strengths and drawbacks of the different structural theories for conceptual modeling.

The empirical assessment was conducted to get feedback from users of the formalism and evaluate adequacy and usability. More specifically, we collect data about the completeness, intuitiveness, experienced complexity of the formalism, and the impact on language design. Furthermore, we investigate dependencies between skills of a language engineer and the experienced complexity, as well as dependencies between the skills and the experienced difficulty to learn MetaMorph. We also gather information about the time needed for formalization as the possibility for supporting the use of the formalism. The results are encouraging and allow for an adjusted research agenda for future improvements of METAMORPH and assisting tools. To conclude, we summarize some lessons learned from the empirical assessment and all case studies conducted so far.

The proposed formalism METAMORPH aims at closing the gap and providing a suitable tool for the phase of *formalization* in the AMME lifecycle of modeling methods and building a bridge between the informal method design and the platform-specific method implementation. By providing a generic and complete formalism like the one proposed, we enable an investigation of the class of formalized modeling languages as a subclass of all formal languages. Furthermore, being complete and unambiguous, the formalization of modeling languages can serve as the single source of specification. This specification can then be translated to arbitrary metamodeling platforms. This is a step towards automatizing the AMME lifecycle of modeling-method engineering.

In the current version, the METAMORPH formalism is subject to several limitations. Firstly, the definition indeed comprises solely the most common concepts of a modeling language, i.e., object types, binary relation types, attributes (single- and multi-valued), constraints, and specialization of object types. Advanced concepts like n-ary relations or parthood are not yet covered. We are permanently working on the enhancement of the formalism and have already proved the integrability of such concepts on the example of specialization on the model level in the form of the power type pattern [16].

Secondly, the formal foundation of many-sorted model theory complies properly with conceptual modeling languages but indeed implicates a higher level of complexity in comparison to, e.g., set theory. Many modeling practitioners are not familiar with this advanced topic in mathematical logic. For this reason, we collected data in the empirical evaluation on the most suitable training material to adjust to the needs of the users. On the other hand, several formalization tasks could be automated as the formalized modeling language or model is usually meant to be consumed by a computer, not by a human. The creation of a prototype of a translator to (metamodel) platform-specific code is on our research agenda.

Thirdly, although functions to alter and work on models are a crucial point for model value, METAMORPH and all other formalisms known to the authors do not admit the specification of algorithms on models. This is an essential shortcoming and one of the research projects receiving special attention on the future agenda. It was initially contemplated as an outlook in [16].

A final point on the research agenda is the potential of logic- and model theory for contributing concepts and methods beneficial for conceptual modeling. We intend to investigate the subclass of conceptual modeling languages in the class of formal languages as defined in logic and approach well-known research topics in conceptual modeling with new tools from logical model theory.

References

1. ADOxx.org: ADOxx Metamodelling Platform, https://www.adoxx.org/live/home
2. 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

3. 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
4. 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). https://doi.org/10.1109/HICSS.2014.422
5. 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 (1989)
6. 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
7. Burger, E.: Flexible Views for View-Based Model-Driven Development. KIT Scientific Publishing, Karlsruhe (2014). https://doi.org/10.5445/KSP/1000043437
8. 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
9. Chang, C.C., Keisler, H.J.: Model Theory, 3rd edn. North-Holland, Amsterdam (1990)
10. Chen, P.: The entity-relationship model—toward a unified view of data. ACM Trans. Database Syst. (TODS) **1**(1), 9–36 (1976). https://doi.org/10.1145/320434.320440
11. Choe, Y., Lee, M.: Algebraic method to model secure IoT. In: Domain-Specific Conceptual Modeling: Concepts, Methods and Tools pp. 335–355 (Jul 2016). https://doi.org/10.1007/978-3-319-39417-6_15
12. 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). https://doi.org/10.1007/978-3-642-28279-9_12
13. 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
14. 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
15. Döller, V.: M2FOL: a formal modeling language for metamodels. In: Bork, D., Grabis, J. (eds.) The Practice of Enterprise Modeling. PoEM 2020. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63479-7_8
16. Döller, V.: Formalizing the four-layer metamodeling stack with MetaMorph: potential and benefits. Softw. Syst. Model. **21**(4), 1411–1435 (2022). https://doi.org/10.1007/s10270-022-00986-2
17. Döller, V., Götz, S.: Tree diagrams and unit squares 4.0: digitizing stochastic classes with the didactic modeling tool ProVis. In: Karagiannis, D., Lee, M., Hinkelmann, K., Utz, W. (eds.) Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools, pp. 481–501. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-93547-4_21
18. 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
19. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer-Verlag, Berlin Heidelberg (2006). https://doi.org/10.1007/3-540-31188-2
20. Enderton, H.B.: A Mathematical Introduction To Logic, 2nd edn. Harcourt/Academic Press, San Diego (2001)
21. 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
22. 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). https://doi.org/10.1007/978-3-642-36654-3_6
23. 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). https://doi.org/10.3233/FAIA200002
24. Guizzardi, G.: Ontological foundations for structural conceptual models. Ph.D. thesis, University of Twente. Wiley, New York (Oct 2005)
25. Guizzardi, G.: Ontology-based evaluation and design of visual conceptual modeling languages. In: Domain Engineering, pp. 317–347, Springer, Berlin, Heidelberg (2013)
26. Heckel, R., Taentzer, G.: Graph Transformation for Software Engineers. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43916-3
27. Henderson-Sellers, B.: On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29825-7
28. Henderson-Sellers, B.: Why philosophize; Why not just model? In: Johannesson, P., Lee, M.L., Liddle, S.W., Opdahl, A.L., Pastor López, Ó. (eds.) Conceptual Modeling. ER 2015, pp. 3–17. Springer International Publishing, Cham (2015)
29. Henderson-Sellers, B., Ralyté, J., Ågerfalk, P.J., Rossi, M.: Situational Method Engineering, vol. 1. Springer, Berlin Heidelberg (2014). https://doi.org/10.1007/978-3-642-41467-1/COVER
30. 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
31. 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). https://doi.org/10.1007/11768869_14
32. 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). https://doi.org/10.1007/978-3-319-73459-0_1
33. Karagiannis, D., Bork, D., Utz, W.: Metamodels as a conceptual structure: some semantical and syntactical operations. In: The Art of Structuring, pp. 75–86. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-06234-7_8
34. 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

35. Karagiannis, D., Kühn, H.: Metamodelling platforms. In: Proceedings of the Third International Conference on E-Commerce and Web Technologies. p. 182. Springer (2002)
36. Karagiannis, D., Mayr, H.C., Mylopoulos, J. (eds.): Domain-Specific Conceptual Modeling: Concepts, Methods and Tools. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-39417-6
37. Karagiannis, D., Lee, M., Hinkelmann, K., Utz, W. (eds.): Domain-Specific Conceptual Modeling: Concepts, Methods and ADOxx Tools. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-93547-4
38. Kaschek, R.: 20 years after: what in fact is a model? Enterp. Model. Inf. Syst. Archit. (EMISAJ) **13**, 28–34 (2018)
39. 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
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. Mayr, H.C., Thalheim, B.: The triptych of conceptual modeling. Softw. Syst. Model. (2020). https://doi.org/10.1007/S10270-020-00836-Z
42. MetaCase: MetaEdit+ Domain-Specific Modeling environment, www.metacase.com/products.html
43. Muck, C., Palkovits-Rauter, S.: Conceptualizing design thinking artefacts: the Scene2Model storyboard approach. In: Domain-Specific Conceptual Modeling pp. 567–587 (2022). https://doi.org/10.1007/978-3-030-93547-4_25
44. Mylopoulos, J.: Conceptual Modelling and Telos. Conceptual Modelling, Databases, and CASE: An Integrated View of Information System Development pp. 49–68 (1992)
45. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: representing knowledge about information systems. ACM Trans. Inf. Syst. (TOIS) **8**(4), 325–362 (1990)
46. Olivé, A.: Conceptual Modeling of Information Systems. Springer-Verlag, Berlin Heidelberg (2007). https://doi.org/10.1007/978-3-540-39390-0
47. OMiLAB NPO: OMiLAB: Open Models Initiative Laboratory (2022), http://www.omilab.org
48. 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 (2013). https://doi.org/10.1007/978-3-642-41924-9_9
49. 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. Bus. Inf. Syst. Eng. **60**, 69–80 (2018)
50. Semeráth, O., Barta, Á., Horváth, Á., Szatmári, Z., Varró, D.: Formal validation of domain-specific languages with derived features and well-formedness constraints. Softw. Syst. Model. **16**(2), 357–392 (2017). https://doi.org/10.1007/s10270-015-0485-x
51. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, 6th edn. McGraw-Hill, New York (2011)
52. Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G.: Metamodelling: State of the Art and Research Challenges, pp. 57–76. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16277-0_3
53. Stachowiak, H.: Allgemeine Modelltheorie. Springer, Cham (1973)
54. Sumereder, A., Dokken, T.: Model-based guide toward digitization in digital business ecosystems. In: Domain-Specific Conceptual Modeling pp. 411–433 (2022). https://doi.org/10.1007/978-3-030-93547-4_18
55. 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). https://doi.org/10.1007/978-3-642-15865-0_17
56. Thalheim, B.: Conceptual model notions-a matter of controversy: conceptual modelling and its lacunas. Enterp. Model. Inf. Syst. Archit. (EMISAJ) **13**, 9–27 (2018)
57. Thalheim, B.: Conceptual models and their foundations. In: International Conference on Model and Data Engineering. pp. 123–139. Springer (2019)
58. Utz, W.: Design metamodels for domain-specific modelling methods using conceptual structures. 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. 47–60. http://CEUR-WS.org (2018)
59. Weisemöller, I., Schürr, A.: Formal definition of MOF 2.0 metamodel components and composition. In: Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M. (eds.) Model Driven Engineering Languages and Systems, pp. 386–400. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
60. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer-Verlag, Berlin Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2

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.

Dimitris Karagiannis holds a full professor position for Business Informatics at the University of Vienna since 1993, leading the Research Group Knowledge Engineering (http://www.dke.univie.ac.at). He received his Ph.D. degree from the Technical University of Berlin in 1987. The same year he joined the Research Institute for Application-oriented Knowledge Processing in Ulm as division head for "Enterprise Information Systems." Prof. Karagiannis holds an honorary professorship from the Babes-Bolyai University in Cluj-Napoca, Romania. His research interests include metamodelling, knowledge engineering, business process management, enterprise architecture management, and artificial intelligence. The industrial application of his metamodelling research was demonstrated within the BOC Group (http://www.boc-group.com), a European software and consulting company founded in 1995. In parallel, scientific applications of his research are applied in the Open Models Laboratory-OMiLAB, http://www.omilab.org, an open collaborative environment for modeling method engineering, which he has established and is currently leading, located in Berlin.

Wilfrid Utz is one of the managing directors of OMiLAB NPO, the non-profit organization headquartered in Berlin supporting the conceptual modeling community organized around emerging topics with respect to domain-specific conceptual modeling. Wilfrid completed his Ph.D. thesis in 2020 at the University of Vienna in the field of metamodel design and knowledge representation using conceptual structures. He has been involved in numerous international research and innovation projects and gained experience in the field of modeling method conceptualization, design, and implementation of modeling tools using the open ADOxx metamodeling platform (http://www.adoxx.org).