A Category System for Data Properties

Hermann Bense[1], Bernhard Thalheim[2], and Thomas Zeh[3]

[1] bense.com Digital Publishing Company GmbH, Schwarze-Brüder-Str. 1, 44137 Dortmund, Germany, hb@bense.com *
[2] Christian-Albrechts-Universität, Institut für Informatik, Olshausenstr. 40, 24098 Kiel, Germany, bernhard.thalheim@email.uni-kiel.de **
[3] Ernst-Schröder-Zentrum für Begriffliche Wissensverarbeitung e.V., TU Darmstadt, Fachbereich Mathematik, Schlossgartenstr. 7, 64289 Darmstadt, Germany, thomas.zeh@t-online.de ***

Abstract. Purpose: Research in knowledge and ontology engineering is experiencing an ongoing effort to extend Conceptual Modeling (CM) to Meta-Level Modeling (MLM). The goal is to achieve greater expressiveness, lower memory requirements and less complex models with a minimal set of modeling elements. **Methodology**: In this paper we present an extended approach to MLM. It is based on a category system of four different types of Data Properties (DP). With the O4Top ontology blueprint we propose a schema that provides a foundation for both conceptual and meta-level modeling. It supports the formalization of structural hierarchies for classes, relationship types and attributes. **Findings**: The proposed category system of data properties meets the requirements of meta-level modeling. Instead of using potency to control the length of the instantiation chains, the data property type determines how far an attribute value can be propagated through an inheritance hierarchy. This supports the streamlining of ontologies and their terminologies. **Originality**: To our best knowledge no other approach for metal-level modeling is founded on a category system of data properties. The naming conventions for ontological elements encode element types with little syntactic overhead and serve to disambiguate the names of the modeling elements.

Keywords: Conceptual Modeling, Meta-Level Modeling, Data Properties, Potency, Deep Instantiation, Power Types, Value Assignment Propagation

1 Introduction

Research in knowledge and ontology engineering is experiencing an ongoing effort to extend Conceptual Modeling (CM) to Meta-Level Modeling (MLM). The goal is to achieve greater expressiveness, lower memory requirements and less complex models with a minimal set of modeling elements. To accomplish their task, ontologies often need to

* https://orcid.org/0000-0002-2562-224X
** https://orcid.org/0000-0002-7909-7786
*** https://orcid.org/0009-0008-9432-8192

represent beside classes (e.g., the class `Car`) and their instances (e.g., the Aston Martin driven by James Bond in Goldfinger) types of classes (e.g., `Car_Model` Aston Martin DB5). MLM is essentially about the interplay of classes and class types and how to assign values to attributes (DPs) of classes. The problem is that after merging a class `Car` with its powertype `Car_Model`, powertype attributes become class attributes. For example, the `Car_Model` attributes `producedUnits` and `YearOfDesign` become attributes of the `Car` class. As a result, it is necessary to introduce rules that govern the inheritance of attributes and values to subclasses and instances. For example, it should not be possible to inherit the `YearOfDesign` of the car model to car instances. These rules should be easy to learn and to master. Often, MLM methods introduce annotations within classes such as potency that determine how and to what extent attributes and their values are inherited, or to what extend attribute-value pairs are propaged down an instantiation chain. Thus, the goal of our study is to provide an MLM methodology that is more expressive while allowing for streamlined modeling.

Multi-Level Modeling, also known as hierarchical modeling, is commonly used in statistical analysis and machine learning to analyze data that is structured hierarchically or nested. Examples include students within courses, patients within hospitals, or repeated measurements taken over time on the same individuals. Meta-Level Modeling (MLM) refers to modeling at a higher level of abstraction. It is commonly used in computer science and software development, especially in Model-Driven Engineering (MDE). Both approaches deal with hierarchical structures, but in different contexts, namely, multi-level modeling in the context of data analysis and meta-level modeling in the context of model abstraction. And, both methods aim at improving decision-making, either by analyzing complex data or by providing robust model structures. In this paper, the focus is primarily on Meta-Level Modeling.

The key research question is: What instantiation rules and guidelines must be applied to Data Properties (DP) when moving from Conceptual Modeling (CM) to Meta-Level Modeling (MLM). How does this affect the modeling style of the ontologists? How can existing CM ontologies benefit from this, and what are the impacts?

The rest of the paper is organized as follows: In the Related Work section 2, we discuss what approaches for Meta-Level Modeling (MLM) already exist and what shortcomings they have. The Preliminaries section

3 introduces the basic nomenclature we use and explains the main features of our minimal ontological top schema O4Top. Then, in the Data Property Types section 4, we introduce four different types of data properties that we think are essential to fulfill the requirements of MLM. In the Examples section 5, we first show how a typical conceptual model is transformed into a meta-level model and how powertypes can be subsumed under their base types. Second, we discuss, how MLM can also be applied to Object Properties (OP) and n-ary relations. In the Discussion section 6, we analyze to what extent our methodology contributes to fulfilling the criteria for Meta-Level Modeling, and how metadata can be managed more systematically. We also discuss the savings potential of our method. In addition, we discuss how the multi-level features postulated by Fonseca et al. [1] are satisfied. Finally, in the Conclusion section 7, we summarize the results and advantages of our approach.

2 Related Work

Meta-Level Modeling (MLM) is about the instantiation of classes from classes in such a way that one class can be viewed as an abstraction of the other at a higher level. For example, a regularity attribute such as `producedUnits` can be aggregated up the `product` inheritance hierarchy. Another aspect of MLM is to model pairs of classes such as `(Car, Car_Model)` where the first class is the base class in an application domain and the second class is the model of the first. In this case, the second class is called the powertype of the first class. In the following, we take typical examples from the literature and discuss this aspect together with the methods of classification as well as shallow and deep instantiation. We find one of the early discussions of powertypes in the paper by Odell [2]. Also, Guizzardi et al. discuss "What's in a Powertype?" [3]. The article by Carvalho and Almeida [4] describes how the powertype `MobilePhoneModel` can be modeled using so-called regularity attributes. However, in their approach, the classes `MobilePhone` and `MobilePhoneModel` cannot be merged, because then each physical `phone` would also have a `launchDate`. Atkinson and Kühne [5] argue that the shallow instantiation approach cannot solve the problems of multiple classification and replication of concepts. The authors add the feature potency to each model element. In each instantiation step, the potency is reduced by 1. However, this requires that the entities to be modeled so that they can be divided into logical levels, each of which must be assigned a meaning. This introduces the potential for layer mistakes, especially in ontologies that change frequently during development. The subdivision of base classes is also often done using

terms such as type, category, family, group, order, subordinate, and so on. For example, Neumayr et al. ([6] on page 9, in Figure 3), model the base classes `Book` and `Car` as subclasses of `ProductCategory` and assign it a classification level of 3. To model that `marketLaunch` must be instantiated at the brand level and `mileage` must be be instantiated by objects at the physical entity level, the authors claim on page 22 that it is necessary to use explicit constraints. Since the authors do not specify the constraints, the question of how this can be achieved is left open. We also see the potential for layer mistakes in taking this approach. Guizzardi et al. [3] also discuss deep instantiation. The powertype class `BirdSpecies` and the base class `Bird` are modelled in two parallel branches of a class hierarchy. We identify a modeling deficit in which the properties of all other species should have to be modeled redundantly as in `BirdSpecies`, or should be moved to a higher class `Species`. We consider the graph rewriting rules on page 6 of [7] to be an important contribution to reducing the complexity of ontology models. Another meta-level model for the biological domain by Batista et al. [8] exemplifies categorization and the use of powertypes. Again, here the question arises, whether the set of rules and axioms for creating multi-level structures listed on page 18 could be reduced by an alternative modeling method. The approach of Bense and Humm [9] proposes to satisfy the instance and type character by two different inheritance mechanisms of properties. These are `BroderInheritedProperty` based on `skos:broader` and `TypeInheritedProperty` based on `rdfs:class`. A case not covered by this is when a modeler wants to express that all `polar bears` have `white fur`. Bense already discusses in [10] three different types of data properties for Meta-Level Modeling (MLM). In comparison, we show in our paper that it is necessary to introduce a fourth type called Transparent Data Property (TDP). Fonseca et al. [1] provide a summary of multi-level approaches and their capabilities according to a list of eight multi-level features. They also discuss MLM features such as durability, mutability and potency, e.g., 'In DeepJava, the potency of an element denotes the maximum depth of its instantiation chain, or how many times a type can be instantiated' or 'However, the only mechanism available for relating features across levels is potency, which is limited to defining how deep a feature is present in an instantiation chain'. In our study we provide an easier to handle method compared with using potency. Attributes that can be instantiated only once in an instantiation chain are considered to be single-potency elements. The authors also discuss the 'multi-level phenomenon called deep instantiation, where the attributes of a higher-order type affect entities at lower levels'. As an example, they mention

that when lions are assigned the feature `warmblooded = T`, this also applies to all particular lions. For the desired functionality that an attribute-value assignments in a class propagates down an instantiation chain to all of the particular instances of those classes we will introduce the so-called datatype Transparent Data Property (TDP). Within the Dublin Core Metadata Initiative[4], metadata is defined in the Dublin Core Metadata Element Set. Are data properties like `Title`, `Publisher` and `Description` really always metadata, or are they application domain specific? It needs to be clarified how to achieve flexibility in modeling so that such attributes can be used to represent either knowledge domain specific information or meta information about the domain model, i.e., information about the model of the domain model.

3 Preliminaries

The three dimensions Generalization, Aggregation and Particularization[5] shown in Figure 1 span what could be considered a kind of knowledge space. Figure 1 shows the layers and dimensions of an ontology $\mathcal{O}$. The set of classes is $\mathbf{C}$ = {C1, ..., C6}, the set of data properties is $\mathcal{A}$ = {A1, ..., A6} and the set of particulars is $\mathcal{E}$ = {P1, ..., P7). The class hierarchy ($\mathcal{C}$, $\leq_\wedge$) is organized along the Generalization dimension. Particulars p $\in \mathcal{E}$ lying in the Particulars Layer (M_0 = PL) are connected to their classes in the Schema Layer (M_1 = SL) by an assertion `(p, »pof, C)`. So, particulars are instantiated along the Particularization dimension. Be $\leq_\wedge$ and $\leq_\Diamond$ non-strict, acyclic, transitive ordering relations for classes and object properties. In the following, »subClassOf and »subOpOf have the same meaning as SubClassOf and SubObjectPropertyOf in OWL, and »subModelOf will be used as a subobject property of »subClassOf. We define class and OP hierarchies (see figure 2) using principal ideals (see Ganter and Wille, [11]) as follows: Be $x \leq_\wedge y \Leftrightarrow x, y \in \mathbf{C}$: (x, »subClassOf, y) $\vee$ (x, »subModelOf, y) such that for $0 < i < j \leq n$ it holds $C_i \leq_\wedge C_j$. Then the class hierarchy ($\mathbf{C}$, $\leq_\wedge$) has the set of classes $\therefore C := \{c \in \mathbf{C} \mid c \leq_\wedge C\}$. The maximum level of ($\mathbf{C}$, $\leq_\wedge$) is the maximum distance of an element from the root of the hierarchy. Therefore, in Figure 1 the class hierarchy has n = 4 levels and minLevel = 0 and maxLevel = 3. Note, that we only distinguish between two layers SL and PL for each concrete domain ontology, which will be complemented by a third abstraction layer

[4] https://www.w3.org/2001/09/rdfprimer/rdf-primer-20021108.html#dublincore

[5] For the instantiation of entities on the Particulars Layer (PL) we prefer the term particularization because instantiation can also take place between classes on the Schema Layer (SL).

of the O4Top blueprint schema in the next section. However, the number of levels in an inheritance hierarchy is not limited. In analogy to a class hierarchy, a hierarchy of Object Properties (OP) is defined as $(\mathbf{R}\ ,\ \leq_{\Diamond})$ and has the set of OPs $\therefore$R := $\{r \in \mathbf{R} \mid r \leq_{\Diamond} R\}$ with $x \leq_{\Diamond} y \Leftrightarrow x, y \in \mathbf{R}$: (x, »subOpOf, y). In general, the extension of the OP ◊R is defined as $\underline{\Diamond R} = \{(x, y) \mid (x,\ \text{»R}\ , y)\}$[6].

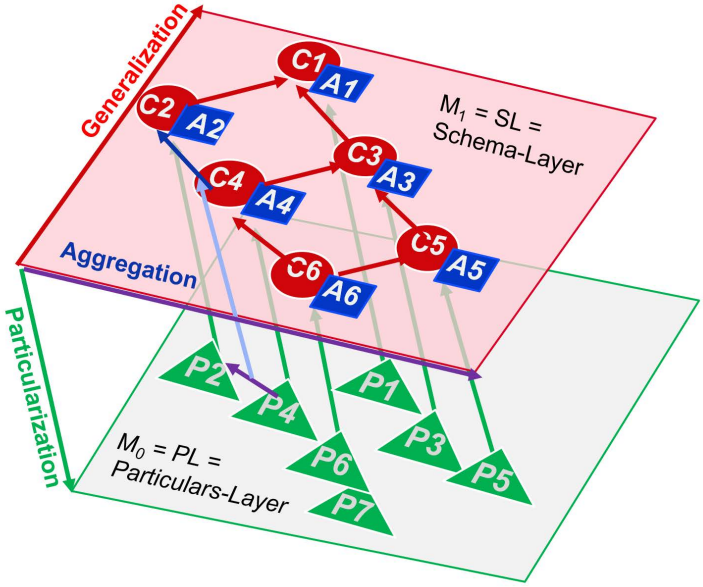

Fig. 1 Ontology Layers

Data Properties (DP) represent intrinsic features of classes and Object Properties (OP) and have a type adt $\in \mathcal{ADT}$ = {:String, :Integer, :Decimal, :Float, :Boolean, :Date, :Time, ...}. We think of intrinsic features as internal properties of things. They characterize and classify those things independently of how those things are related to other things in an application domain. A class C aggregates declarations for a data property `A` in the form of the Data Property Definition `DPD = (C, A, adt)` along the Aggregation dimension of the SL layer. In a DPD the class `C` plays the role of the domain and the `adt` plays the role of the range.

Object Properties (OP) / Binary Relationship Types represent extrinsic features of universals, i.e., how things are related to other things. This can,

[6] Note that the underline characterizes a set and in this case the underline is essential for the definition of an extension.

for example be merenomy / parthood relations or family relationships. Object Property Instantiations (OPI) can take place between particulars `P` and `Q` in the PL layer. An OPI takes the form `(P, »op, Q)`. The blue arrow from `C4` to `C2` in figure 1 indicates an `opd = (C4, ◊op, C2)` while the purple arrow indicates an `opi = (P4, »op, P2)`. In a Object Property Definition (OPD) `(C, ◊op, D)` the class `C` plays the role of the domain class and D the role of the range class.

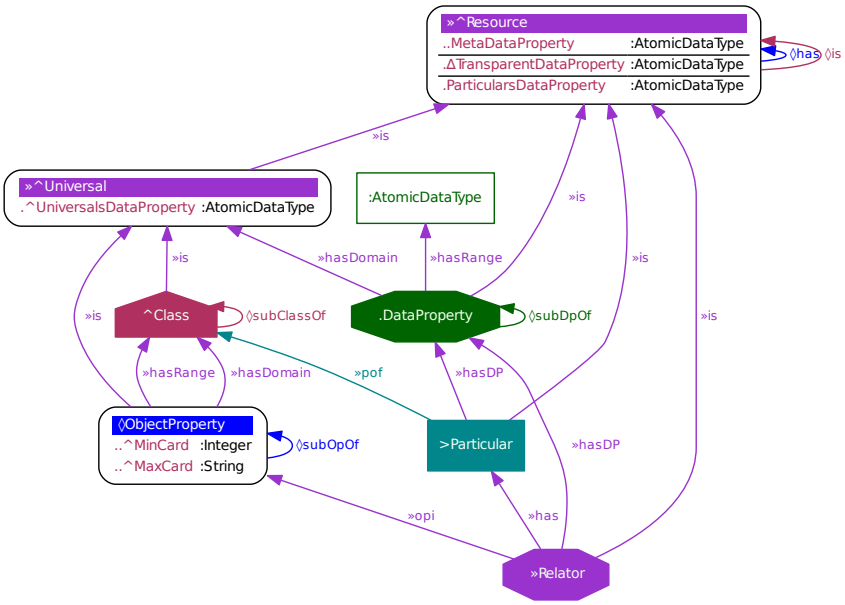

Fig. 2 O4Top Schema

The O4Top schema in figure 2 shows a so-called OntoGraph with a minimal set of entity types needed for Conceptual Modeling (CM) and Meta-Level Modelling (MLM). O4 is an abbreviation for Ontology4 and refers to our ontology development methodology and tools based on four different types of data properties[7]. Please note that the O4Top schema does not constitute an independent ontology layer. However, all elements of the O4Top schema, except for the example elements >Particular and »Relator, can also be used in any M1 schema layer of ontologies. For disambiguation purposes we use prefixes to name the elements. This is analogous to the use of prefixes by Carvalho and Almeida [4], where, for example, the prefix `instances` in `instancesScreenSize` implicitly denotes a regularity attribute. We are aware that using the prefixes introduces

[7] https://o4.studio

additional learning overhead, but on the other hand this is achieved with little syntactic overhead. Thus, we accept this tradeoff in favor of introducing more complex annotations with additional definitions.

- The most general modeling element is »^Resource. It subsumes with »is the universals ^Class and ◊ObjectProperty as well as .DataProperty. This is the basis for maintaining the hierarchies of the three elements with the transitive object properties ◊subClassOf, ◊subOpOf and ◊subDpOf. It also contains the schemas for the four different data property declarations required for the instantiation down the hierarchies. »^Universal is the specialization of »^Resource that provides the schema for defining data properties of type `.^UniversalsDataProperty`, which can only be defined in universals.
- Classes (prefixed with circumflex ^) and Object Properties (OP) (prefixed with diamond ◊ for OP definitions and double greater » for OP instances[8]) are universals. Examples are ^Person, ^Product, ^Book and ◊hasMotor, »subClassOf, »hasRange.
- Data Properties (DP) (prefixed with dot .) can be assigned to universals and are defined with atomic data types from the set $\mathcal{ADT}$ between a class or OP (»hasDomain) and an adt $\in \mathcal{ADT}$ (»hasRange). Examples are .serialNr, .milage, .IMEI, .OrcID, .DUNSnr, .ISBN. A Data Property Definition (DPD), e.g., for .serialNr with (.serialNr, »hasDomain, ^Car) ∧ (.serialNr, »hasRange, :String) is equivalent to its short definition (^Car, .serialNr, :String)
- Object Properties (OP) are always defined between a domain class (»hasDomain) and a range class (»hasRange), e.g., (◊hasMotor, »hasDomain, ^Car) ∧ (◊hasMotor, »hasRange, ^Motor) which is equivalent to the short definition (^Car, ◊hasMotor, ^Motor). In an OP name the prefix ◊ refers to an Object Property Definition (OPD) while the prefix » refers to an Object Property Instance (OPI). The OPIs named with »has, »hasDomain, »hasRange and »hasDP can be considered as instantiation of ◊has, while ◊subClassOf, ◊subOpOf and ◊subDpOf are derived from ◊is with ◊subClassOf ⊆ ◊is, ◊subOpOf ⊆ ◊is, and ◊subDpOf ⊆ ◊is.
- Particulars (prefixed with >) are associated with their classes by »pof (particular of), e.g., (>BTs-Porsche, »pof, ^Porsche911CarreraS).
- Relators are associated with their Object Property Definition (OPD) using »opi, for example, (»hasAdvisor#3, »opi, ◊hasAdvisor) in figure 5. They are used to model binary and n-ary relationships.

[8] An OP instance relates two particulars

Why do we consider the set of modeling elements in the O4Top schema to be minimal? A modeler cannot omit any of the elements without losing the ability to model essential facts. Classes represent the universals of an application domain. Object Properties establish binary relationships between classes and their instances. Data Properties represent properties of classes, binary relationships and their instances and are the basis for classification. Atomic data types characterize the nature of data properties. Thus, none of the modeling elements can be omitted or replaced by any of the others. On the other hand, experience from countless modeling projects has shown that there is no need to introduce additional types of modeling elements.

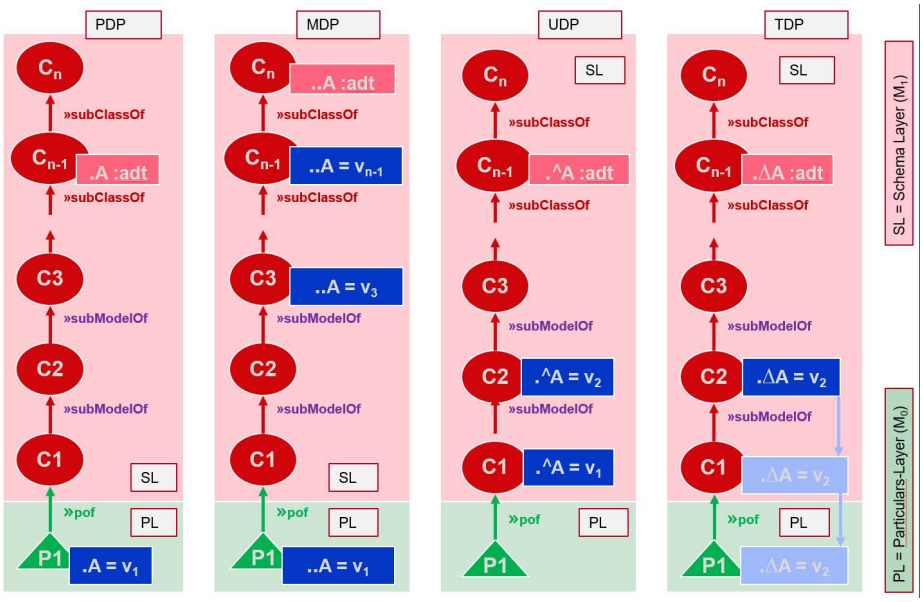

Fig. 3 Data Property Types

4 Data Property Types

Typically, in classical Conceptual Modeling (CM), Data Properties (DP) are only used to assign values to attributes of particulars that have been previously declared in a class. In Meta-Level Modeling (MLM), data property values can also be assigned within classes. In this chapter we examine the types of data properties needed to meet the requirements of MLM. This also results in restrictions for their value assignment. We will therefore refer to the classic data properties from CM as Particulars Data Properties (PDP). Meta Data Properties (MDP) allow entities to be annotated with metadata at the Schema Level (SL) and at the Particulars

Level (PL). Universal Data Properties (UDP) can only be assigned values in Schema Level (SL) entities. With Transparent Data Properties (TDP), a value assignment can be made within a class in such a way that the assignment of that attribute value is propagated from that class down the hierarchy to the particulars in the PL. In the following, each of the data property types is discussed in detail, as shown in figure 3. For the following let $1 \leq i \leq j \leq n$. In each class hierarchy (C_i , $\leq_\wedge$) a definition DPD = (C, A, adt) for a data property `A` is only allowed for one class `C`. For example, let us consider the domain classes `^Person` and `^Product` as the roots of subclass hierarchy branches under the top class `^Class`. Then, a data property such as `weight` can be defined only once in common superclass of the two classes. In the following, we generally assume for the roots C_n of domain subclass hierarchies that they are subclasses of the top class `^Class` (see figure 2) where $(C_n$, »subClassOf, ^Class).

4.1 Particulars Data Property (PDP)

The purpose of PDPs is to assert domain-specific facts exclusively in the Particulars Layer (PL). PDP names are prefixed with a dot (.). Examples of PDPs are .serialNr, .fonNr, .personId, .URI. A definition for a PDP attribute .A has the form DPD_P:= (Cj, .A, :adt). For any given P with (P, »pof, C_i) and $C_i \leq_\wedge$ Cj a value for attribute .A can be assigned with (P, .A, v) based on its definition $DPD_P = (C_j, .A, :adt)$.

4.2 Meta Data Property (MDP)

The purpose of MDPs is to assert non-domain-specific metadata to knowledge subjects, such as any kind of authoring and administrative information. Names are prefixed with dot-dot (..). Examples for MDPs are ..createdBy, ..updated, ..checkedDate, ..VersionNr, ..DevTime, ..DevCost. An MDP definition for an MDP attribute ..A takes the form DPD_M:= (C_j, ..A, :adt). For any particular P with (P, »pof, C_i) and $C_i \leq_\wedge$ Cj a value for attribute ..A can be assigned with (P, ..A, v) based on the $DPD_M = (C_j, ..A, :adt)$. For any class $C_i \leq_\wedge C_j$ a value v_i for attribute ..A can be assigned with (C_j, ..A, v_i) based on $DPD_M = (C_j, ..A, :adt)$.

4.3 Universals Data Property (UDP)

The purpose of UDPs is to assert data property values to universals (classes and object properties) that are **not** specific to particulars of those universals. A major motivation for the introduction of UDPs is that they make it possible to combine the powertype of a class such as `Car_Model` with the base type `Car` into one class `Car`. This is achieved by turning the

PDP attributes of the `Car_Model` powertype such as `.qtySold` into UDP attributes such as `.^producedUnits` in the Car class. UDPs are derived DPs. Typically, resultant or range or cardinality properties such as sum, average, min, max etc. can be represented with UDPs. Names are prefixed with a period-circumflex (.^). Examples for UDPs are `.^producedUnits`, `.^qtySold`, `.^avgLifeTime`, `.^maxSpeed` etc. A UDP Definition for a UDP attribute .^A has the form DPD_U:= (Cj, .^A, :adt). Particulars of any class from $\therefore C_n$ may not instantiate UDPs. For any class $C_i \leq_\wedge C_j$ a value v_i for attribute .^A can be assigned with (C_j, .^A, v_i) based on DPD_U = (C_j, .^A, :adt).

4.4 Transparent Data Property (TDP)

The purpose of TDPs is to assign values to data properties of classes where the pair (.Δattribute, value) propagates down through any entity of the hierarchy to the entities of the Particulars Layer (PL). We also refer to this method as Value Assignment Propagation (VAP). Thus, TDPs allow for deep instantiation as mentioned in [1]. Typically, those properties can be represented by TDPs that are characteristic of classes and their subclasses as well as all of their particulars. TDP names are prefixed with period-delta (.Δ). Examples for TDPs are .Δwarmblooded, .ΔModelName, .ΔBatteryPowered, and so on. A definition for a TDP .ΔA has the form DPD_T:= (C_j, .ΔA , :adt). For any class $C_i \leq_\wedge C_j$ a value v_i for the attribute .ΔA can be assigned with (C_j, .ΔA, v_i) based on DPD_T = (C_j, .ΔA, :adt). This has a huge potential for savings, since pairs of (.ΔA, v_i) do not need to be explicitly assigned in classes and particulars below the first assignment. The reason is that they can be entailed down the instantiation chain. For performance reasons, a modeler might still choose to store the attribute-value pairs redundantly.

4.5 Potency, Durability and Mutability

Carvalho and Almeida [4] give the following definitions for durability and mutability: 'The durability of an attribute indicates how far the attribute spans in an instantiation tree. The mutability of an attribute defines how often the attribute value can be changed in the instantiation tree'. We also find a definition of the potency of an element, which denotes the maximum depth of its instantiation chain. In this sense, its meaning is close to that of durability. Based on figure 3, we consider an instantiation tree to be the subclass hierarchy branch (C_n, $\leq_\wedge$) starting from the root class C_n and including all particulars of (C_n, $\leq_\wedge$). For a more precise definition of the span of durability we define as the starting point of the

span the level where the data property is defined. According to these definitions, we determine the values of durability and mutability for each of the four types of data properties:

- For PDPs, the durability is 1 because PDPs span from a class to one of its particulars. The mutability is 0 because PDPs do not instantiate on the Schema Layer (SL).
- For MDPs, the maximum durability is n, because MDPs span down through the instantiation chain to the Particulars Layer (PL). The mutability is n.
- For UDPs, the maximum durability is n - 1, because UDPs span down through the instantiation chain but not down into the PL. The mutability is n - 1.
- For TDPs, the maximum durability is n, because the TDPs span down through the instantiation chain to the PL. The mutability is n.

5 Examples

The Ontograph visualization [12] in figure 4 represents an extended modeling of the MObjects example by Neumayr et al. from [6], page 9, figure 3. In their example, potency-based deep instantiation is used. The consequence of their use of potency is that 'Additional abstraction levels for some domain concept cannot be introduced without requiring global model changes'. In our modeling we show that such global changes can be avoided: The ^Product class hierarchy contains at the top the classes that are related through »subClassOf. Starting with the class ^Porsche911, the vehicle domain contains a subhierarchy in which the classes are linked by »subModelOf. The following applies: ◊subModelOf ⊆ ◊subClassOf ⊆ ◊is. The inheritance of the Data Properties (DP) applies to the entire ^Product class hierarchy including the models. In classic Conceptual Modeling (CM), the class `^Car` and the associated class `^Car_Model` would be modeled in two parallel branches of the class hierarchy. `^Car_Model` is then referred to as the PowerType (PT) of the base class `^Car` with `^Car_Model = PT (^Car)`. Using our method of Meta-Level Modeling (MLM), these two classes were merged into the single class `^Car`. The prerequisite for this is that the definition and instantiation of the data properties comply with the restrictions described in the section Data Property Types 4. It is always possible to return from MLM to CM, for example, by separating two classes `^Car` and `^Car_Model` from the one class `^Car`. The two classes must then be connected by OPs such as `(^Car_Model, ◊isCarModelOf, ^Car)`. Now we give concrete examples for the different types of data properties.

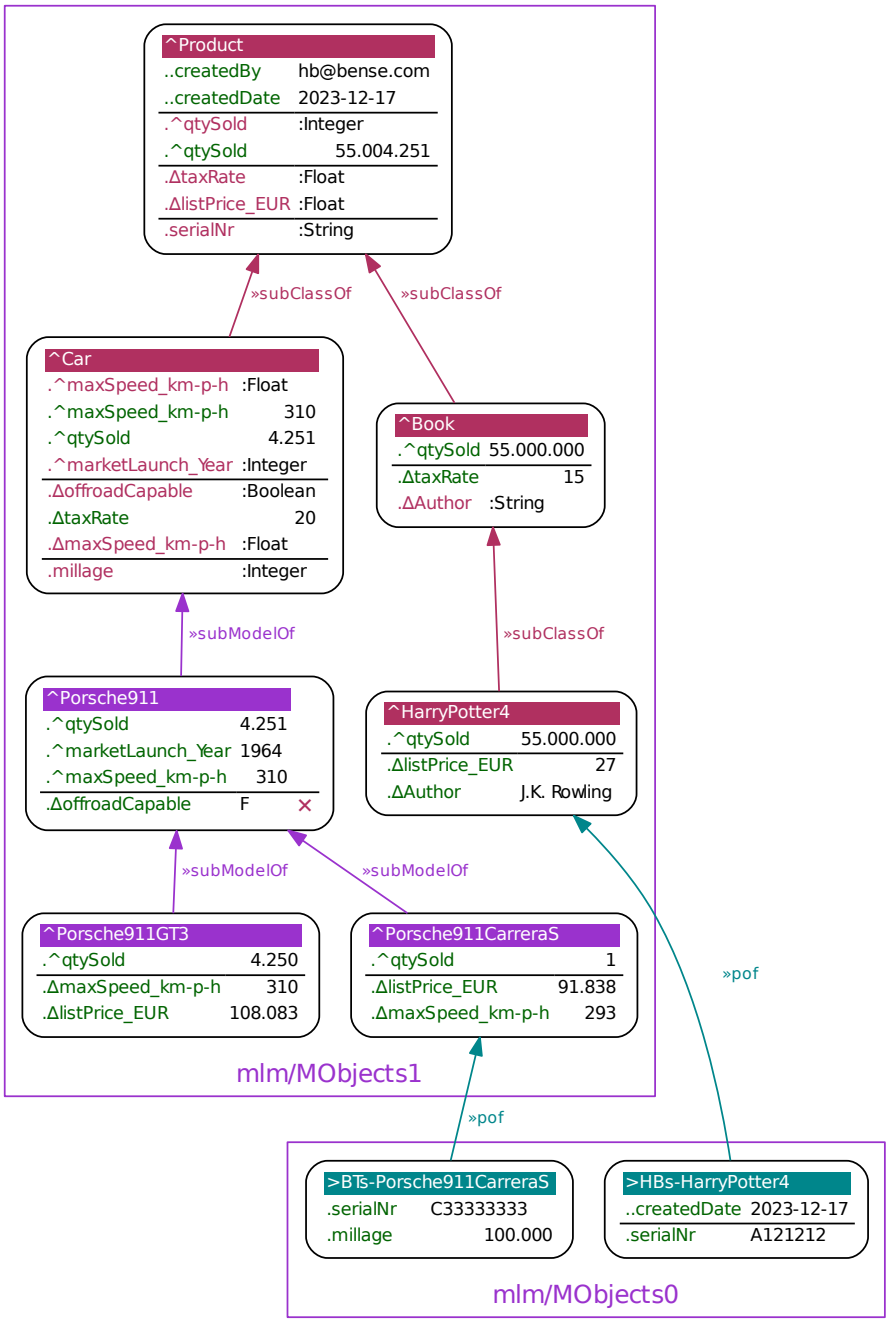

Fig. 4 The MObjects Product Model

– Particulars Data Property (PDP): For example, a PDP can be assigned with (^Product, .serialNr, :String), (>HBs-HarryPotter4, »pof, ^HarryPotter4), and (>HBs-HarryPotter4, .serialNr, A121212).
– Meta-Data Property (MDP): With (»^Resource, ..createdDate, :Date) and (^Product, ..updated, 2013-12-17) it is modeled that ^Product was updated on 2013-12-17. To the best of our knowledge none of the other MLM approaches provide a similar method for maintaining meta-information for knowledge entities.
– Universals Data Property (UDP): The assignments (^Product, .^qtySold, :Integer), (^Product, .^qtySold, 55.004.250), (^Porsche911GT3, .^qtySold, 4.250), and (^HarryPotter4, .^qtySold, 55.000.000) add the value for .^qtySold to the value of .^qtySold on the next level up. Therefore, .^qtySold can be considered as a resultant attribute. The minimum durability of .^qtySold is zero because it has been both defined and assigned in the ^Product class.
– Transparent Data Property (TDP): With (^Product, .ΔtaxRate, :Float), (^Car, .ΔtaxRate, 20), and (^Book, .ΔtaxRate, 15) the attribute-value pairs are propagated through the entire hierarchy. I.e., ^HarryPotter4 also has the .ΔtaxRate = 15 as well as each individual copy such as >HBs-HarryPotter4. The same is true for the TDP .ΔoffroadCapable. So, with (^Car, .ΔoffroadCapable, :Boolean) and (^Porsche, .ΔoffroadCapable, F) the feature (.ΔoffroadCapable, F) can be entailed for each Porsche model as well as for each of its particulars. The value of the UDP .^maxSpeed_km-p-h is the maximum speed of the TDP ΔmaxSpeed_km-p-h of the car models of the next lower level, e.g., it holds that (^Porsche911,.^maxSpeed_km-p-h, 310) because 310 is the maximum({310, 293}) of the models ^Porsche911GT3 and ^Porsche911CarreraS. All of this can add up to huge savings, because in a large ontology, hundreds, thousands or millions of explicit assignments can be saved. In the example, the durability of ΔtaxRate is 4 because it has been defined in the class ^Product and value assignments of ΔtaxRate propagate down to the Particulars Layer (PL).

The graph in figure 5 represents the information in table 2 of [13]. To quote Nguyen et al. [13]: "A singleton property is defined as a property instance identifying the unique relationship between the subject and the object". The relationship between `>ProfessorA` and the universities `>University1` and `>University2` is represented by the two relators `»worksFor#1` and `»worksFor#2`. So, we interpret the two relators as singletons in the above sense. Both are instantiated from the ◊`worksFor`

Object Property (OP), which is a ◊subOpOf the OP ◊TimeRelation. The latter relation is denoted by the »opi OP. Thus, it is possible to model any number of working relationships for >ProfessorA, to each of which the PDPs .from and .to can be assigned. The underlying Object Property Definition (OPD) is (^Professor, ◊worksFor, ^University), which is equivalent to (◊worksFor, »hasDomain, ^Professor) ∧ (◊worksFor, »hasRange, ^University). In the example, ◊worksFor represents a binary relationship type. Note that »worksFor#1 is an OP instance of the OP ◊worksFor, which means that it is also an element of ◊TimeRelation and ◊ObjectProperty. But (>ProfessorA, »TimeRelation, >University1) and (>ProfessorA, »ObjectProperty, >University1) cannot be entailed.

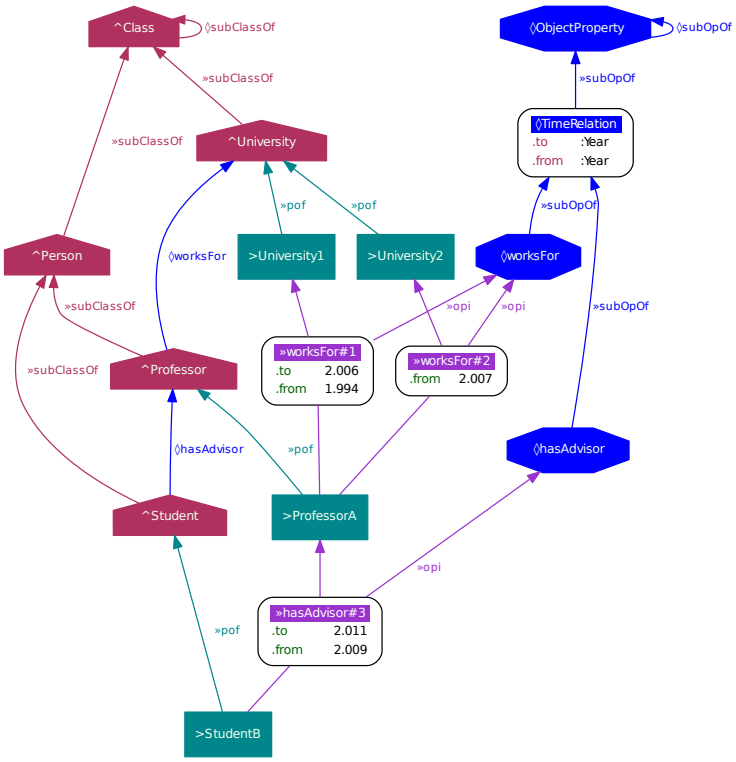

Fig. 5 Attributes of Object Properties

An n-ary relationship type can be modeled by having one domain class, e.g. ^Transport and n range classes, e.g., ^Agent, ^Product, ^Route, ^Instrument, ^Period. This can be interpreted as ^Transport being a reification of n binary relationship types. Other typical examples of

n-ary relations are ^Surgery, ^Employment, ^Marriage, ^ClassEnrollment and so on. In [14] it is described how arbitrary complex state of affairs can be modeled with so-called Cascaded Role Sets (CRS). Additionally, the OPs in figure 5 form an OP hierarchy where for their extensions it holds that ◊hasAdvisor ⊆ ◊TimeRelation, ◊worksFor ⊆ ◊TimeRelation, and ◊TimeRelation ⊆ ◊ObjectProperty. In future work we will elaborate on our insight that the instantiation for MDPs, UDPs and TDPs can also be applied for OPs in analogy to classes. Currently, we are also investigating candidates for two additional attribute types in order to capture the possibility of supporting Value Assignment Propagation (VAP) for universals (..ΔMetaTransparentDataProperty), and another one for the representation of meta information on universal (..^MetaUniversalsDataProperty). The example in the figure 5 can be extended by a modeling option for partitioning classes that is only possible through the use of Transparent Data Properties (TDPs). This is done by declaring the .ΔGender attribute in the ^.Gender class. In the subclasses ^.female and ^.male the values female/weiblich and male/männlich are assigned for English and German. Particulars then get the corresponding value via Value Assignment Propagation (VAP), e.g. for >StudentB the value male can be entailed as assigned in ^.male. Thus, compared to [1], the explicit modeling of powertypes such as PersonType, PersonTypeByGender and EmployeeType can be simplified.

6 Discussion

Compared to the modeling of Carvalho and Almeida in [4], Figure 14, we consider our approach to be more expressive in that it allows to merge the base class MobilePhone and the powertype class MobilePhoneModel while preserving the intended semantics. With regularity attributes alone, MobilePhoneModel cannot be subsumed under MobilePhone, because then each particular phone would also have a launchDate. Also, we do not need to add and maintain a feature such as potency for every data property. The way in which potency is used by Frank [15] has the consequence that non-trivial continuous updates of the potency values are required when extending the class hierarchies. Our methodology dictates which instantiation rule is to apply by using the prefixes in the DP names. Also, in our approach, as in deep instantiation, universals can have both instance and type character simultaneously. This is comparable to clabjects as presented by Atkinson and Kühne [5]. In the paper by Neumayr et al. ([6], page 9, figure 3) the base classes Book and Car are modeled as subclasses of ProductCategory. We would merge the

`ProductCategory` with the class `Product` and avoid a forced division into classification levels. Also for the same reason, compared to the modeling in [6], page 11, figure 4 we see no need to explicitly model ProductModel to be `classifiedAs` ProductCategory.

6.1 Meta Data

If one wants to model not only the application domain but also the model of the domain model, then this is done by Meta Data Properties (MDPs). As a convention, we agree that MDPs are defined in the O4Top schema (figure 2). This is similar to the Dublin Core Metadata Initiative. In the adopted example `AnnotationAssertion(a:createdBy a:Dog "Seth MacFarlane")` from OWL2[9] the `createdBy` attribute can be considere a typical MDP and we would place its definition in `»^Resource` with (`»^Resource, ..createdBy, :String`). Thus, any ontology element in the instantiation chain down from `»^Resource` can take an MDP assertion for `..createdBy`.

6.2 Attribute Propagation

Dahchour et al. in [16], page 8, give examples for the three different types T1, T2 and T3 of attribute propagation. T1 propagation corresponds to our method Value Assignment Propagation (VAP) by Transparent Data Properties (TDP). However, what is missing in [16] as an instantiation method is the ability to specify that values for attributes in classes may not propagate into particulars as provided by Universals Data Properties (UDP). This prevents the authors from defining resultant properties (as defined in [7], page 2) such as `.^qtySold` or `.^maxSpeed_km-p-h`, which would lead to erroneous value propagation if additional metaclasses were not introduced. We map their T0 propagation to MDPs in our methodology. We have also shown that metaclass constructs such as `instance-type` or `instance-instance-type` are not needed to capture the semantics of generic relationship types, contrary to what is shown on page 14 of [16]. This also means that by using TDPs the various approaches shown in figure 12 on page 14 for materialization with meta classes are not required.

6.3 Multi-Level Features

Following the list of multi-level features F1, ..., F8 presented by Fonseca et al. in [1], we discuss how our approach meets their requirements:

– F1: Figure 4 shows with the »subClassOf, »subModelOf hierarchy how we can represent entities of multiple classification levels.

[9] https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Metamodeling

– F2: The number of classification levels is not limited.
– F3: The guiding principles for organizing models are given by the criteria for selecting the appropriate type of data property as described in section 4.
– F4: Our most generic modeling entity is »^Resource as shown in the O4Top schema in figure 2. It allows universals such as classes and object properties to inherit data properties and to propagate pairs of DPs and values down a hierarchy.
– F5: The rules governing the instantiation of related types are given by the DP restrictions in section 4. In figure 4 we have shown, how the powertype `^Car_Model` can be subsumed under the base type `^Car`.
– F6: The feature of supporting the representation of properties (data properties and relationships) of types as well as the assignment of values to their instances was discussed in the section Examples 5.
– F7: Figure 3 explains the relationship between DPs of entities in different classification levels. There, the »subModelOf classification level is aligned under the »subClassOf classification level where $\underline{\Diamond\text{subModelOf}} \subseteq \underline{\Diamond\text{subClassOf}}$ due to »subModelOf $\leq_{\Diamond}$ »subClassOf.
– F8: The representation of domain relations between entities of different classification levels is supported without durability, mutability or potency.

6.4 Type Cascading Supposition (TCS)

If `^Car_Model` is the PowerType (PT) of `^Car` then we can write `PT(^Car) = ^Car_Model`. The question then arises, is there also a `^Car_Model` powertype, e.g. `^Car_Model_Model = PT(^Car_Model) = PT(PT(^Car))`? We think not, because what attributes could `PT(^Car_Model)` have? In the example shown in figure 4, we used the Object Property (OP) `»subModelOf` to subsume car models under the base class car. In analogy the same would work for `(Bird, Bird_Species)` with `»subSpeciesOf`, for `(Person, Person_Role)` with `»subRoleOf`, for `(Product, Product_Category)` with `»subCategoryOf` and so on. As a modeling guideline, we can deduce that the name used to specify the powertype can be used to find the name for the subsumption OP. In general, we create the OP Name `»subSpecOf` for `Class_Spec`. Then with (`»subSpecOf`, `»supOpOf`, `»subClassOf`) and (`»subClassOf`, `»supOpOf`, `»is`) it holds in general $\underline{\Diamond\texttt{»subSpecOf}} \subseteq \underline{\Diamond\texttt{»subClassOf}} \subseteq \underline{\Diamond\texttt{»is}}$. The example in figure 4 also shows that, compared to other instantiation methods such as deep or dual deep instantiation ([17], [3]), we do not need to annotate with a feature such as potency.

7 Conclusion

We have discussed how to extend Conceptual Modeling (CM) to appropriately provide the features needed for Meta-Level Modeling (MLM). Many other MLM approaches require features such as potency, durability and mutability. This places a heavy burden on the modelers, since any change in an inheritance hierarchy may require the renumbering of potencies. We circumvent this problem by introducing simple naming conventions for the four required data property types. We have provided criteria for selecting the appropriate data property type. As shown in the Examples section 5, it is possible to implement powertypes and their subsumption as well as the attribution of binary and n-ary relation types. However, MLM makes higher demands on the modeler, because he has to understand and master additional instantiation mechanisms. But the advantages are greater expressiveness, lower memory requirements, less complex models and as a side effect, layer mistakes can be reduced or even eliminated. So, as a general guideline for modelers, we suggest, to prefer MLM, since the backdoor to return to CM is always open.

References

1. Fonseca CM, Almeida JPA, and Giancarlo Guizzardi VAC. Multi-level conceptual modeling: Theory, language and application. Data Knowledge Engineering 134(1):101894 2021. DOI: 10.1016/j.datak.2021.101894. Available from: https://www.researchgate.net/publication/351567433_Multi-level_conceptual_modeling_Theory_language_and_application
2. Odell JJ. Power Types. Journal of Object-Oriented Programming, Volume 7(2) 1994 :8–12
3. Guizzardi G, Almeida JPA, Guarino N, and Carvalho VA. Towards an Ontological Analysis of Powertypes. International Workshop on Formal Ontologies for Artificial Intelligence (FOFAI) 2015. Available from: https://www.researchgate.net/publication/279178175_Towards_an_Ontological_Analysis_of_Powertypes
4. Carvalho VA and Almeida JPA. Toward a Well-Founded Theory for Multi-Level Conceptual Modeling. 2016. Available from: https://nemo.inf.ufes.br/wp-content/papercite-data/pdf/toward_a_well_founded_theory_for_multi_level_conceptual_modeling_2016.pdf
5. Atkinson C and Kühne T. The Essence of Multilevel Metamodeling. International Conference on the Unified Modeling Language 2001 :19–33. Available from: https://link.springer.com/chapter/10.1007/3-540-45441-1_3
6. Neumayr B, Schrefl M, and Thalheim B. Modeling Techniques for Multi-Level Abstraction. 2008. DOI: 10.1007/978-3-642-17505-34. Available from: https://www.researchgate.net/publication/221024521_Modeling_Techniques_for_Multi-level_Abstraction

7. Guizzardi G, Figueiredo G, and Poels MMHG. Ontology-Based Model Abstraction. IEEE Thirteen International Conference on Research Challenges in Information Science, Brussels 2019. DOI: 10.1109/RCIS.2019.8876971. Available from: https://www.researchgate.net/publication/332290797_Ontology-Based_Model_Abstraction
8. Batistaa JO, Almeida JPA, Zambona E, and Guizzardi G. Ontologically Correct Taxonomies by Construction. Data Knowledge Engineering 2022
9. Bense H and Humm B. An Extensible Approach to Multi-Level Ontology Modelling. KMIS 2021, 13th International Conference on Knowledge Management and Information Systems 2021
10. Bense HJ. Streamlining Conceptual Modeling. Kohei Arai (edt.), Proceedings of the 2023 Intelligent Systems Conference (IntelliSys), Amsterdam, The Netherlands, Springer Nature, Lecture Notes in Networks and Systems 822 2023. ISBN: 978-3-031-47721-8:326–44. Available from: https://link.springer.com/chapter/10.1007/978-3-031-47721-8_22
11. Ganter B and Wille R. Formal Concept Analysis - Mathematical Foundations. 2nd Edition, Springer Berlin Heidelberg 2024. ISBN: 978-3-031-63421-5
12. Bense H. The Unique Predication of Knowledge Elements and their Visualization and Factorization in Ontology Engineering. Kutz O, Garbacz P (eds.), Proceedings of the Eighth International Conference (FOIS 2014), Rio de Janeiro, Brazil, Sept. 22-25 2014 :241–50. DOI: 10.3233/978-1-61499-438-1-251. Available from: https://ebooks.iospress.nl/publication/37972
13. Nguyen V, Bodenreider O, Thirunarayan K, Gang Fu EB, Rosinach ŃuQ, Laura I. Furlong MD, and Sheth.Authors A. On Reasoning with RDF Statements aboutStatements using Singleton Property Triples. 2015. Available from: https://arxiv.org/pdf/1509.04513.pdf
14. Bense H. Modeling Ontology Design Patterns with Cascaded Role Sets. The 9th Joint Ontology Workshop (JOWO 2023), Sherbrooke, Quebec, Canada 2023. Available from: https://ceur-ws.org/Vol-3637/paper1.pdf
15. Frank U. Multi-level modeling: cornerstones of a rationale. Software and Systems Modeling 21 2022 :451–80. Available from: https://doi.org/10.1007/s10270-021-00955-1
16. Dahchour M, Pirotte A, and Zimányi E. Materialization and its Metaclass Implementation. IEEE Transactions on Knowledge and Data Engineering 14(5) 2002 :1078–94. DOI: 10.1109/TKDE.2002.1033775. Available from: https://www.researchgate.net/publication/3297079_Materialization_and_its_metaclass_implementation
17. Carvalho VA, Almeida JPA, and Guizzardi G. Using a Well-Founded Multi-Level Theory to Support the Analysis and Representation of the Powertype Pattern in Conceptual Modeling. 2011. Available from: http://www.inf.ufes.br/~gguizzardi/caise2016.pdf