Multi-level conceptual modeling: Theory, language and application

Claudenir M. Fonseca [a],[*], João Paulo A. Almeida [b], Giancarlo Guizzardi [a],[c], Victorio A. Carvalho [d]

[a] *Free University of Bozen-Bolzano, Italy*
[b] *Federal University of Espírito Santo, Brazil*
[c] *University of Twente, The Netherlands*
[d] *Federal Institute of Espírito Santo, Brazil*

ARTICLE INFO

ABSTRACT

In many important subject domains, there are central real-world phenomena that span across multiple classification levels. In these subject domains, besides having the traditional type-level domain regularities (classes) that classify multiple concrete instances, we also have higher-order type-level regularities (metaclasses) that classify multiple instances that are themselves types. Multi-Level Modeling aims to address this technical challenge. Despite the advances in this area in the last decade, a number of requirements arising from representation needs in subject domains have not yet been addressed in current modeling approaches. In this paper, we address this issue by proposing an expressive multi-level conceptual modeling language (dubbed ML2). We follow a principled language engineering approach in the design of ML2, constructing its abstract syntax as to reflect a fully axiomatized theory for multi-level modeling (termed MLT*). We show that ML2 enables the expression of a number of multi-level modeling scenarios that cannot be currently expressed in the existing multi-level modeling languages. A textual syntax for ML2 is provided with an implementation in Xtext. We discuss how the formal theory influences the language in two aspects: (i) by providing rigorous justification for the language's syntactic rules, which follow MLT* theorems and (ii) by forming the basis for model simulation and verification. We show that the language can reveal problems in multi-level taxonomic structures, using Wikidata fragments to demonstrate the language's practical relevance.

1. Introduction

A class (or type) is a ubiquitous notion in modern conceptual modeling approaches and is used in a conceptual model to establish invariant features of the entities in a domain of interest. Often, subject domains are conceptualized with entities stratified into a rigid two-level structure: a level of classes and a level of individuals, which instantiate these classes. In many subject domains, however, classes themselves may also be subject to categorization, resulting in classes of classes (or metaclasses). For instance, consider the domain of biological taxonomies [1–3]. In this domain, a given organism is classified into taxa (such as, e.g., Animal, Mammal, Carnivoran, Lion), each of which is classified by a biological taxonomic rank (e.g., Kingdom, Class, Order, Species). Thus, to represent the knowledge underlying this domain, one needs to represent entities at different (but nonetheless related) classification levels. For example, Cecil (the lion killed in the Hwange National Park in Zimbabwe in 2015) is an instance of Lion, which is an instance of Species. Species, in its turn, is an instance of Taxonomic Rank (like Kingdom, Class, Order). Other examples of multiple classification levels come from domains such as software development [4] and product types [5].
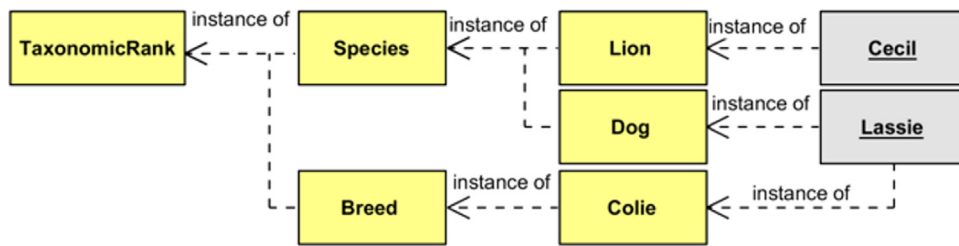
Fig. 1. Four levels of classification in a representation of a biological domain.

In the last decade, the importance of phenomena involving multiple levels of classification and the limitations of the fixed two-level scheme have motivated the development of a number of modeling approaches under the banner of "Multi-Level Modeling" (e.g., [5–8]). These approaches embody conceptual notions that are key to the representation of multi-level models, such as the existence of entities that are simultaneously types and instances (classes and objects), the iterated application of instantiation across an arbitrary number of (meta)levels, the possibility of defining and assigning values to attributes at the various type levels, etc.

Despite these advances, a number of requirements arising from representation needs in subject domains have not yet been addressed in most of the modeling approaches currently available. For example, many approaches do not support domain relations between elements of different classification levels. Some others impose rigid constraints on the organization of elements into strictly stratified levels, effectively obstructing the representation of genuine domain models.

These issues are addressed in the design of an expressive multi-level conceptual modeling language which we call ML2 (Multi-Level Modeling Language). The language is aimed at multi-level (domain) conceptual modeling and is intended to cover a comprehensive set of multi-level domains. We follow a principled approach in the design of ML2, defining its abstract syntax to reflect a formal theory for multi-level modeling which we developed previously (MLT*, reported in [9]). We propose a textual syntax for ML2, which is supported by a featured Xtext-based editor in Eclipse. ML2 enables the expression of a number of multi-level modeling scenarios that cannot be currently expressed in the existing multi-level modeling languages. Further, we show how ML2 incorporates rules to prevent the construction of unsound multi-level models. These rules follow theorems in MLT*, providing rigorous justification for the language design. In order to show the practical relevance of the language's rules, we select fragments of multi-level taxonomies in the Wikidata knowledge base[1] and represent them in ML2. We show that the language is able to reveal issues that are not ruled out by Wikidata's representation scheme and have passed unnoticed by curators of numerous multi-level taxonomies in Wikidata.

This paper is further structured as follows: Section 2 presents the MLT* theory and its formalization in first-order logics, forming the semantic foundation of ML2; Section 3 presents ML2's abstract and concrete syntax; Section 4 presents MLT*'s theorems and derived rules showing how they drive the design of ML2 rules and the language's support for model validation and simulation; Section 5 shows how ML2's syntactic rules (corresponding to MLT* theorems) reveal issues in Wikidata taxonomies; Section 6 discusses related work, comparing ML2 to existing multi-level techniques; finally, Section 7 presents our final considerations. The paper is a revised and extended version of [10]. The original paper did not cover the role of the formalization in the language's design. This paper also extends the original paper in the use of Wikidata content to demonstrate the usefulness of ML2's rules in a large-scale scenario, effectively addressing a significant problem in Wikidata's level-blind support that some of us had originally identified in [11]. There is also extended coverage of related work.

2. MLT*: The Multi-Level Theory

We employ the MLT* Multi-Level Theory [9] which incorporates and extends MLT [2]. The theory was conceived as a reference top-level ontology for types in multi-level conceptual modeling. It provides basic concepts and patterns to articulate domains that require multiple levels of classification [2].

Our use of MLT* as a semantic foundation for ML2 is motivated by the successful application of MLT in diverse modeling initiatives. For example, in [12], MLT has been used to account for a redesign of the UML's powertype pattern support, exposing limitations in its original support and improving its use for a broader range of multi-level modeling scenarios. A similar effort used MLT to reveal limitations in Semantic Web support for multi-level modeling [3]. In [13], MLT was combined with the Unified Foundational Ontology (UFO) [14] to support the representation of higher-order types in ontologically well-founded conceptual models. In addition to serving as a basis for the redesign of representation strategies, the theory has also been used in different application areas. For example, in [15], MLT was used as a basis for the modeling of organizational structures in different levels of generality. In [16], the authors report the use of MLT to support information modeling in a real data-integration project from a heavy vehicle manufacturer (Scania CV AB).

---

[1] Wikidata comprises a model for structuring data for projects such as Wikipedia, Wikivoyage, Wiktionary, Wikisource, among others (https://www.wikidata.org/).
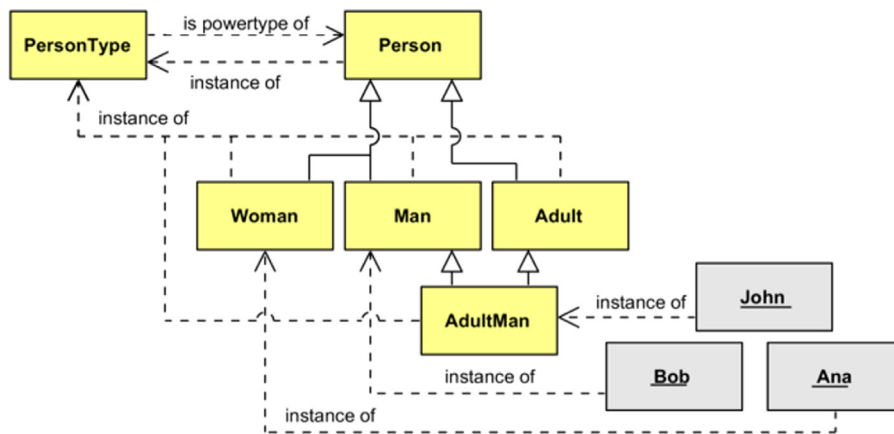
Fig. 2. PersonType and its instances.

2.1. Elements of the theory

MLT* is concerned with types, their instances, and the relations between them. Types are conceived in the theory as predicative entities (e.g. "Person", "Organization", "Product") that can possibly be applied to a multitude of entities (including types as well). If a type t applies to an entity e then it is said that e is an *instance of* t. In contrast, individuals are entities that cannot possibly have instances (e.g. "John", an apple, my cellphone). In the philosophical literature, types are said to be repeatable, while instances are non-repeatable [14].

Since a type can be an instance of another type, it is possible to conceive of chains of instantiations (of any size), in order to represent multiple levels of classification. For instance, Fig. 1 presents an example in the biological domain with four classification levels, from the individuals Cecil and Lassie, until the type TaxonomicRank. Some of the types are both instances and classifiers of other entities, for example, Lion classifies Cecil and is simultaneously an instance of (or is classified by) Species. In the following examples, we use a hybrid notation inspired in both the class and object notations of UML merely for illustrative purposes. We use dashed arrows to represent relations that hold between the elements, with labels to denote the relation that applies (in Fig. 1, the relations depicted are instance of relations).

The theory is built up first by defining common structural relations to support conceptual modeling, starting from specialization between types. Structural relations that reflect variants of the powertype pattern are also included, given the pervasiveness of this pattern in descriptions of multi-level phenomena. Based on Cardelli's notion of powertype [17], we define that t is powertype of t' iff every instance of t is a specialization of t' and all specializations of t' are instances of t. For example, in Fig. 2, PersonType is the powertype of Person, thus every specialization of Person (e.g. Man, Woman, Adult and even Person itself) instantiates it, throughout the specialization hierarchy (e.g. AdultMan is also an instance of PersonType).

In order to address also the notion of powertype introduced by Odell [18], MLT* also includes the so-called categorization relation. A type t categorizes a base type t' iff all instances of t are proper specializations of t'. Differently from Cardelli's powertype, in a categorization, the base type t' is not itself an instance of t. Furthermore, not all possible specializations of t' are instances of t. For instance, as presented in Fig. 3, EmployeeType (with instances Manager and Researcher) categorizes Person, but is not the powertype of Person, since there are specializations of Person that are not instances of EmployeeType (Woman and Man for example).

The theory also defines useful variations of the categorization relation: (i) a type t completely categorizes a type t' iff every instance of t' is instance of at least one instance of t; (ii) a type t disjointly categorizes a type t' iff every instance of t' is instance of at most one instance of t; finally, (iii) t partitions t' iff every instance of t' is instance of exactly one instance of t. In Fig. 3, PersonTypeByGender partitions Person into Man and Woman, and thus each instance of Person is either a Man or a Woman but not both simultaneously.

One can observe that, as presented in Fig. 2, entities in a subject domain can be organized based on their levels. For example, Person and its specializations only classify entities that are individuals (e.g., John, Bob and Ana), while PersonType sits at a higher level, classifying Person and other types. MLT* accounts for this organization of entities into levels using the notion of type order. Types whose instances are individuals are called first-order types. Types whose instances are first-order types are called second-order types. Those types whose extensions are composed of second-order types are called third-order types, and so on.

Since they fall under a strictly stratified scheme, i.e., they only have instances pertaining to the level immediately below, these types are called *ordered types*. The theory explicitly accounts for orders using the notion of *basic type*. A basic type is the most abstract type of its order, i.e., the type whose extension includes all entities in the order immediately below. For example, Individual is the basic type whose extension includes all individuals, 1stOT is the basic type whose extension includes all first-order types, 2ndOT is the one that classifies all second-order types, and so on. Due to this definition, all basic types are related in a chain of powertype relations, as presented in Fig. 4, with every ordered type specializing the basic type of the same order and instantiating the one of
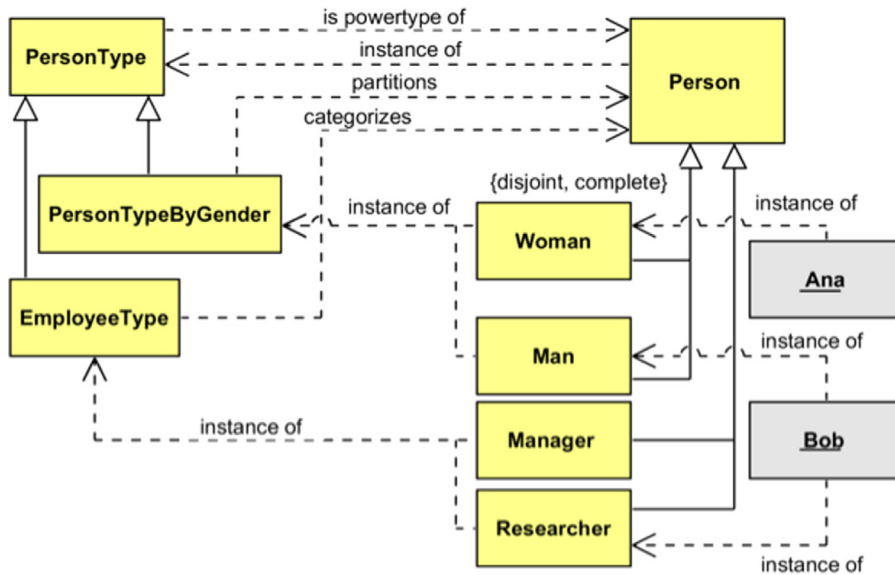
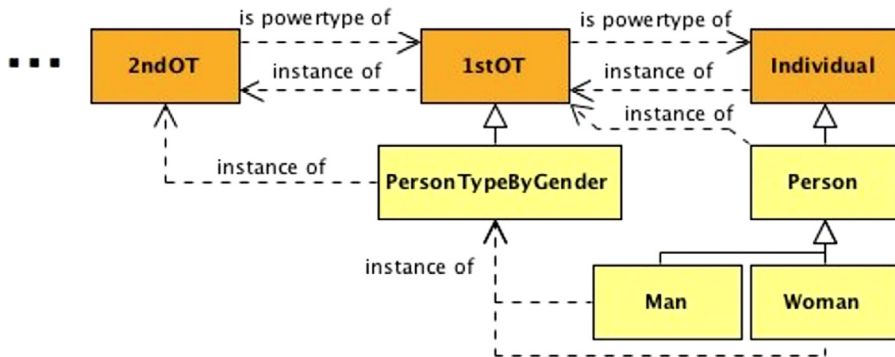**Fig. 3.** Examples of the categorization and partitions relations.

Fig. 4. Example of MLT*'s basic types.

the order above (e.g., Person). The ellipsis in the figure represents that this chain of basic types can be extended as far as demanded, given the entities involved in the subject domain. However, the formalization of MLT* does not necessitate infinite chains of basic types, allowing the description of finite models (see [9] for details).

This stratification into type orders provides for a structure useful to guide modelers in producing sound models. However, this is unable to account for types whose instances do not fall into a unique order. Examples of such types include the notions of Entity [14], Thing [19,20] and Property [21], which are key to a number of comprehensive conceptualizations. In MLT*, types that do not conform to the stratified scheme are termed *orderless types*. Consider the notion of BusinessAsset as an economic resource owned by some enterprise. In this context, we may say that Apple's AppleParkMainBuilding is a business asset as well as its cellphone models, e.g., IPhone5. Note that while the former is an individual, the latter is a first-order type that has individual mobile phones as its instances. Since BusinessAsset has instances in different orders, it is an example of a domain orderless type. Finally, MLT* can also be used to describe its own categories of types resulting in the model shown in Fig. 5.

2.2. Features in multi-level models: Deep instantiation

The theory also accounts for a multi-level phenomenon called *deep instantiation* [22], when the attributes of a higher-order type affect entities at lower levels. For example, whether a species is warmblooded in fact determines whether particular animals of that species are warmblooded. In MLT, this phenomenon is addressed through the so-called *regularity attributes* [2,23]. In the case of a regularity attribute, values defined for a higher-order type (such as second- and third-order types) affect the intension of the instances of the higher-order types. In other words, some attributes of a higher-order type aim at capturing regularities over instances of its instances, constraining the set of possible instances of its instances. Here, we could say that the attribute warmbloodedSpecies of
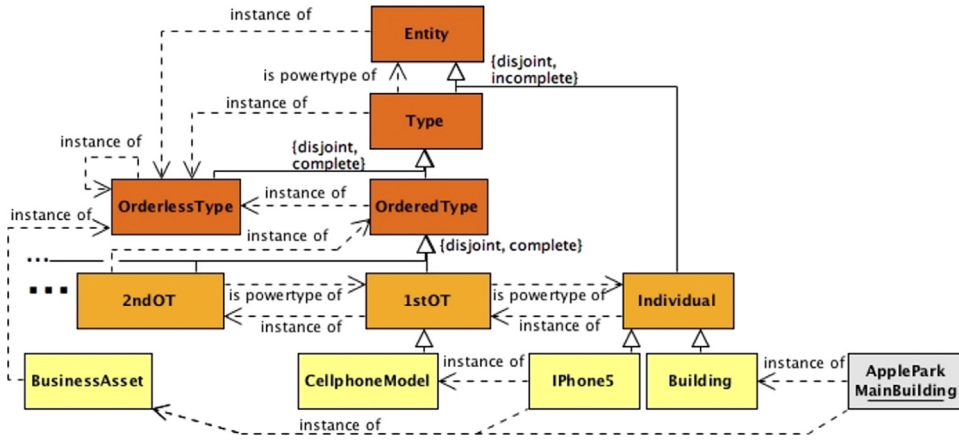
Fig. 5. MLT* basic scheme extended by a domain example.

AnimalSpecies regulates the attribute isWarmblooded of Animal. In the case warmbloodedSpecies is true for an animal species (such as Lion) all its instances (such as Cecil) must have isWarmblooded equals true.

2.3. Axiomatization in first-order logics

MLT* is formalized as an axiomatic theory in first-order logics, quantifying over all possible types and individuals, which together constitute the entities we are interested in (a1). Every entity (type or individual alike) can be related to one or more types through a primitive instanceof relation (or iof, for short). Since types can themselves be related to other types through iof, this enables chains of instantiation of arbitrary lengths. Unlike types, individuals can never play the role of type in instantiation (a2). We assume that the theory is only concerned with types with non-trivially false *intensions*, i.e., with types that have possible instances (a3).

a1 $\quad \forall x(\text{entity}(x))$

a2 $\quad \forall x(\text{individual}(x) \leftrightarrow \neg \exists y(\text{iof}(y, x)))$

a3 $\quad \forall x(\text{type}(x) \leftrightarrow \exists y(\text{iof}(y, x)))$

We establish that types are ultimately grounded on individuals. Thus a super-relation (iof′) is defined including all pairs such that iof(x, y), and also all pairs derived from a chain of pairs connected by iof relations. The transitive (iof′) relation then always leads us from a type to one or more individuals:

a4 $\quad \forall t(\text{type}(t) \rightarrow \exists x(\text{individual}(x) \wedge \text{iof}'(x, t)))$

First-order types are those whose instances are individuals (a5). Second-order types are those instances are first-order types (a6), and so on. Any number of ordered types can be defined in this way [2].

a5 $\quad \forall x(\text{firstordertype}(x) \leftrightarrow \text{type}(x) \wedge \forall y(\text{iof}(y, x) \rightarrow \text{individual}(y)))$

a6 $\quad \forall x(\text{secondordertype}(x) \leftrightarrow \text{type}(x) \wedge \forall y(\text{iof}(y, x) \rightarrow \text{firstordertype}(y)))$

Specialization between types is defined as usual, i.e., a type specializes a supertype whenever all its instances are also instances of the supertype (a7). *Proper specialization* is defined for the cases in which the extension of the specialized type is a proper subset of the extension of the general type (a8).

a7 $\quad \forall t_1, t_2(\text{specializes}(t_1, t_2) \leftrightarrow \text{type}(t_1) \wedge \forall x(\text{iof}(x, t_1) \rightarrow \text{iof}(x, t_2)))$

a8 $\quad \forall t_1, t_2(\text{properSpecializes}(t_1, t_2) \leftrightarrow \text{specializes}(t_1, t_2) \wedge \neg\text{specializes}(t_2, t_1))$

Finally, two types are considered equal iff all their possible instances are the same (i.e., if they are necessarily co-extensional):

a9 $\quad \forall t_1, t_2(\text{type}(t_1) \wedge \text{type}(t_2) \rightarrow (t_1 = t_2 \leftrightarrow \forall x(\text{iof}(x, t_1) \leftrightarrow \text{iof}(x, t_2))))$

Powertypes and variants. Relations between types were defined accounting for different notions of powertype used in the literature, more specifically clarifying and positioning conflicting definitions of Cardelli [17] and Odell [18]. A type t_1 isPowertypeOf a (base) type t_2 iff all instances of t_1 are specializations of t_2 and all possible specializations of t_2 are instances of t_1 (a10). Powertypes in this sense are analogous to powersets. The powerset of a set A is a set that includes as members all subsets of A (including A itself). The categorizes relation between types was defined to reflect Odell's notion of powertype [18]. Differently from Cardelli's, Odell's

definition excludes the base type from the set of instances of the powertype. Further, not all specializations of the base type are required to be instances of the powertype (a11). Odell's definition corresponds more directly to the notion of powertype that was incorporated in UML. Thus, there may be specializations of the base type that are not instances of the categorizing higher-order type. For example, we may define a type named Organism Type by Habitat (with instances Terrestrial Organism and Aquatic Organism) that *categorizes* Organism. Organism Type by Habitat is not a (Cardelli) powertype of Organism since there are specializations of Organism that are not instances of Organism Type by Habitat (e.g. Plant and Golden Eagle) [3].

a10 $\forall t_1, t_2(\text{isPowertypeOf}(t_1, t_2) \leftrightarrow \text{type}(t_1) \land \forall t_3(\text{iof}(t_3, t_1) \leftrightarrow \text{specializes}(t_3, t_2)))$

a11 $\forall t_1, t_2(\text{categorizes}(t_1, t_2) \leftrightarrow \text{type}(t_1) \land \forall t_3(\text{iof}(t_3, t_1) \rightarrow \text{properSpecializes}(t_3, t_2)))$

Specializations of the *categorization* relation were defined in order to capture different scenarios of categorization: *disjoint categorization*, when each instance of the base type is instance of at most one instance of the higher-order type; *complete categorization*, when each instance of the base type is instance of at least one instance of the higher-order type; and *partitioning*, when an instance of the base type is instance of exactly one instance of the higher-order type.[2]

Subordination. A unique feature of MLT* (and hence ML2) is the support for the so-called *subordination* relation. *Subordination* occurs between higher-order types implying proper specializations between their instances i.e., a higher-order type t_1 is subordinate to t_2 iff every instance of t_1 proper specializes an instance of t_2 (a12). A representative application of *subordination* relations occurs in the domain of biological taxonomy for living beings. For example, according to that taxonomy, every instance of *Breed* (e.g. *Collie*) must specialize an instance of *Species* (e.g. *Dog*). Therefore, *Breed isSubordinateTo Species* (See [9] for a full example on the application of MLT* to the domain of biological taxonomy).

a12 $\forall t_1, t_2(\text{isSubordinateTo}(t_1, t_2) \leftrightarrow \text{type}(t_1) \land \forall t_3(\text{iof}(t_3, t_1) \rightarrow \exists t_4(\text{iof}(t_4, t_2)$
         $\land \text{properSpecializes}(t_3, t_4))))$

Constants for MLT's basic scheme.   The entities in MLT's basic scheme are also entities in the domain of quantification of the axiomatic theory. They can be identified by using constants whose interpretation is properly restricted (see (a13), (a14), (a15), which define Individual, 1stOT, and 2nOT respectively, corresponding to the classes in Fig. 4).

a13 $\forall t(t = \text{Individual} \leftrightarrow \forall x(\text{iof}(x, t) \leftrightarrow \text{individual}(x)))$

a14 $\forall t(t = \text{1stOT} \leftrightarrow \forall x(\text{iof}(x, t) \leftrightarrow \text{firstOrderType}(x)))$

a15 $\forall t(t = \text{2ndOT} \leftrightarrow \forall x(\text{iof}(x, t) \leftrightarrow \text{secondOrderType}(x)))$

Ordered and orderless types in general. Finally, as discussed in Section 2.1, the notions of ordered and orderless types are anchored on the notion of "basic type" defined in (a16). A basic type is either the type whose instances are all and only the individuals (Individual) or a type whose instances are all types that specialize another basic type (b_{i-1}) (1stOT, 2nOT, and so on). Note that i is only used to improve the intuition in the definition, and is not formally a variable. This definition forms a chain of basic types of higher-order (without necessitating an infinite chain). The axiom allows for as many orders as required.

a16 $\forall b_i(\text{basicType}(b_i) \leftrightarrow \text{type}(b_i) \land$
         $(\forall x(\text{individual}(x) \leftrightarrow \text{iof}(x, b_i)) \lor$
         $\exists b_{i-1}(\text{basicType}(b_{i-1}) \land \forall t_{i-1}(\text{specializes}(t_{i-1}, b_{i-1}) \leftrightarrow \text{iof}(t_{i-1}, b_i)))))$

An ordered type is a type that specializes a basic type (a17). All other types are orderless types (a18).

a17 $\forall x(\text{orderedType}(x) \leftrightarrow \text{type}(x) \land \exists b(\text{basicType}(b) \land \text{specializes}(x, b)))$

a18 $\forall x(\text{orderlessType}(x) \leftrightarrow \text{type}(x) \land \neg\text{orderedType}(x))$

3. ML2: The multi-level modeling language

ML2 is a textual modeling language for multi-level conceptual models that reflects the concepts and rules of MLT*. MLT* provides to ML2 sound theoretical foundations needed to address the demands of multi-level modeling with a high degree of generality. The development of ML2 has been based on the Xtext framework and provides a featured Eclipse editor, including model validation capabilities and compatibility with EMF-based technologies.[3]

3.1. Modeling multi-level entities

The constructs of ML2 reflect the categories of entities defined by MLT*, as shown in the Ecore model of Fig. 6.
Besides minor terminological differences (with Class representing the notion of type for consistency with EMF terminology, and EntityDeclaration representing entities in general), there are specific constructs for every sort of entity previously presented:

---

[2] Formalization for these cases, which was omitted here, is quite straightforward and can be found in details in [9].
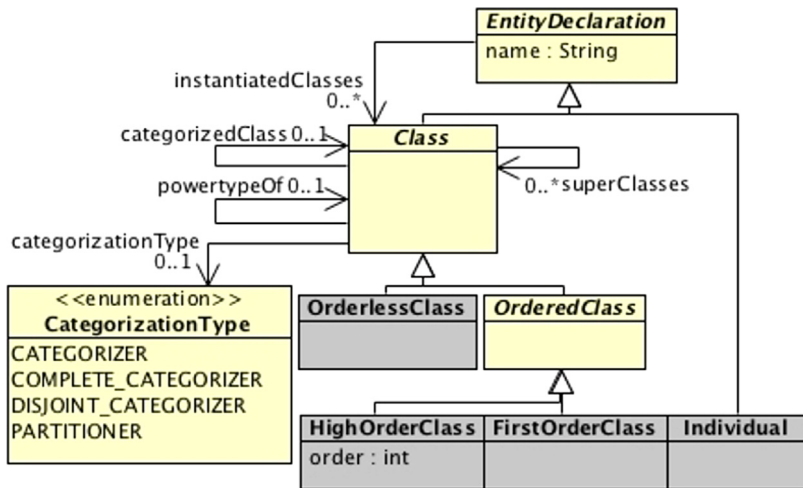
[3] The ML2 editor can be found at https://github.com/nemo-ufes/ML2-Editor.

Fig. 6. Entities and classes in ML2.

Individual representing entities with no instances; FirstOrderClass representing regular classes from the two-level scheme; HighOrderClass representing an ordered class at a certain order; and OrderlessClass representing entities whose extension span across different orders. Instantiation of multiple classes may be declared for all entities, thus including classes and individuals. Specialization (proper) and the other structural relations of the theory are considered for classes. For a class that categorizes another class, a categorization type should be defined to reflect which variant of the categorization relation applies.

The syntax of ML2[4] is inspired in mainstream object-oriented languages. The statements for entity declaration follow a common pattern, varying the available structural relations for each type of entity. Listing 1 revisits some of the examples presented so far, to illustrate ML2's concrete syntax. It includes an orderless class (BusinessAsset) and three second-order classes (namely, CellphoneModel; PersonType which is declared as a Cardelli powertype of first-order class Person; and, EmployeeType which is declared an Odell powertype of Person.) With respect to first-order classes, an instance of CellphoneModel (IPhone5) and two instances of EmployeeType (Manager and Researcher) are provided. An individual (Bob instance of Person and Researcher) is also declared. Note that a namespace mechanism is supported with "modules".

```
1   module example.model {
2     orderless class BusinessAsset;
3
4     order 2 class CellphoneModel;
5     order 2 class PersonType isPowertypeOf Person;
6     order 2 class EmployeeType specializes PersonType
7       categorizes Person;
8
9     class IPhone5 : CellphoneModel, BusinessAsset;
10    class Person : PersonType;
11    class Manager : EmployeeType specializes Person;
12    class Researcher : EmployeeType specializes Person;
13
14    individual Bob : Person, Researcher;
15  }
```

Listing 1: Examples of entity declarations in ML2.

3.2. Features and assignments

Classes contain common features of their instances, while entity declarations contain value assignments for the features of the classes that an entity instantiates. Fig. 7 presents how features are handled in the abstract syntax. ML2 distinguishes features into references and attributes (not unlike Ecore and OWL, for example). A feature has a type, which is a class in the case of references or a datatype in the case of attributes. Datatypes are first-order classes that have as instances particular values. For example, the datatype String is a first-order class that has as instances all well-formed sequences of characters. ML2 supports both user created datatypes and a set of primitive types, namely String, Number and Boolean. The set of primitive types covers a minimal set of data types for conceptual modeling and was inspired in JSON's specification [24].

---

[4] The language's grammar is available at https://github.com/nemo-ufes/ml2-grammar.
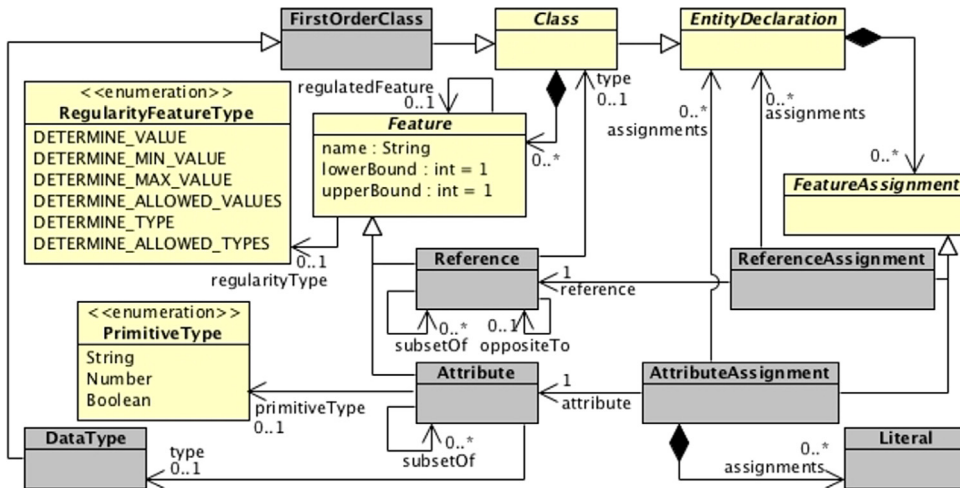
Fig. 7. Features and assignments in ML2.

Listing 2 presents an example of usage of features in an ML2 model. This model expands the one in Listing 1 by explicitly capturing the cross-level reference owns between Enterprise and BusinessAsset and adding some other entities (business assets at different orders such as IPhone5 and AppleParkMainBuilding, which exemplify BusinessAsset being applied as an orderless class). Note that ML2 does not require exhaustive feature assignment, thus intentionally allowing partial representations (see IPhone5 without an assignment for belongsTo). However, cardinality constraints are checked for every assignment that is explicitly captured in a model.

```
1   orderless class BusinessAsset {
2     ref belongsTo : Enterprise isOppositeTo owns
3   };
4
5   class Enterprise {
6     ref owns : [0..*] BusinessAsset isOppositeTo belongsTo
7   };
8   class Building;
9
10  order 2 class CellphoneModel categorizes Cellphone;
11  class Cellphone {
12    screenSize : Number
13  };
14  class IPhone5 : CellphoneModel, BusinessAsset
15    specializes Cellphone
16  {
17    ref belongsTo = Apple
18  };
19
20  individual AppleParkMainBuilding : Building, BusinessAsset;
21  individual Apple : Enterprise {
22    ref owns = { AppleParkMainBuilding, Iphone5 }
23  };
24  individual MyIphone : IPhone5 {
25    screenSize = 4
26  };
```

Listing 2: Examples of features in ML2.

ML2 also accounts for a special kind of feature called *regularity feature* (see [2,23]). This kind of feature has the characteristic of constraining features at a lower level. Consider the previous example of CellphoneModel with an instancesScreenSize feature that represents the specific screen size of a certain model. This feature regulates the screenSize feature of Cellphone, since every cellphone will have the same screen size specified by its respective model. Instances of CellphoneModel such as IPhone5 specialize Cellphone and determine specific value for instancesScreenSize, in this case, 4 inches. Thereby, all instances of IPhone5 have screenSize following the value assigned to instancesScreenSize, i.e., 4 inches. Note that, in order to regulate a feature of Cellphone, the high-order type CellphoneModel must categorize Cellphone, since the regulation of a feature is defined in instances

of the high-order type affecting specializations of the categorized type. Listing 3 presents a modification of Listing 2 illustrating the usage of regularity features.

```
1  order 2 class CellphoneModel categorizes Cellphone {
2    regularity instancesScreenSize : Number
3      determinesValue screenSize
4    regularity ref compatibleProcessorModel : ProcessorModel
5      determinesType installedProcessor
6  };
7
8  class Cellphone {
9    screenSize : Number
10   ref installedProcessor : Processor
11 };
12 class IPhone5 : CellphoneModel specializes Cellphone {
13   instancesScreenSize = 4
14   ref compatibleProcessorModel = A6
15 };
16
17 order 2 class ProcessorModel categorizes Processor;
18
19 class Processor;
20 class A6 : ProcessorModel specializes Processor;
21
22 individual Processor01 : A6;
23 individual MyIPhone : IPhone5 {
24   screenSize = 4
25   ref installedProcessor = Processor01
26 };
```

Listing 3: Examples of regularity features in ML2.

ML2 foresees six types of regularity features. In the case above, values of instancesScreenSize determine the exact value of screenSize. However, a regularity feature can also: determine maximum or minimum values for a number feature (e.g., to model the maximum storage capacity of a cellphone model); determine the set of allowed values for a feature (e.g., to model that a phone model has either 16GB or 32GB of internal storage capacity); or further constrain the type of assignment for a feature, by either determining its type(s) or determining a set of allowed types [14]. The specification of the regularity type can be omitted when the type of regulation does not fit one of the six foreseen types of regulation.

Listing 3 also presents an example in which the regularity reference compatibleProcessorModel of CellphoneModel determines the type of installedProcessor for instances of Cellphone. Since IPhone5 assigns A6 to compatibleProcessorModel, instances of IPhone5 can only have processors that are instances of A6. This is the case of MyIPhone, with Processor01 installed on it. Assignments of regulated features, when present in a ML2 specification, are subject to conformance checks.

4. Derivation of language rules

A key aspect of the design of ML2 concerns its adoption of a clear theoretical foundation. Not only the language's abstract syntax follows the conceptual backbone of the theory (as shown in the metamodel presented in Fig. 6), but further, syntactic rules in the language are directly derived from theorems of the formalized theory, with verification by the ML2 editor. These syntactic rules ensure that multi-level models produced in the language are sound with respect to the theory. The objective is to ensure that syntactically-valid models in ML2 can be given semantics in terms of MLT*, corresponding to a valid model (now in the model-theoretic sense) of the theory.

Section 4.1 presents the rules that apply to the relations between the various elements in MLT* and consequently ML2 (instantiation, specialization, subordination, powertype, and categorization). Section 4.2 focuses specifically on additional rules that can be imposed on the supported powertype pattern variants. Section 4.3 discusses the role of the Alloy formal method [25] in the design of MLT* and in the simulation of ML2 models.[5]

The rules discussed throughout this section are later applied in Section 5 in fragments obtained from Wikidata, exposing modeling problems that can be directly detected with our language.

---

[5] By model simulation here, we mean the formal validation approach provided by Alloy [25] It consists of automatically generating visual representations of model-theoretical instances of a formal specification (in this case, ML2 specifications).

Table 1
Summary of constraints and properties of MLT* relations between types.

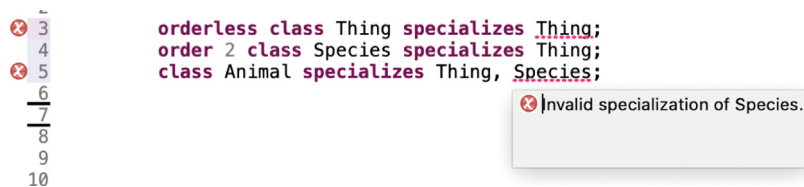| Relation	Domain and Range	Constraint	Properties
specializes(t, t')	orderless:orderless ordered:orderless ordered:ordered	if t and t' are ordered types, they must be at the same type order	Reflexive, antissymetric, transitive
properSpecializes(t, t')	orderless:orderless ordered:orderless ordered:ordered	if t and t' are ordered types, they must be at the same type order	Irreflexive, antissymetric, transitive
isSubordinateTo(t, t')	orderless:orderless ordered:orderless ordered:ordered	t and t' cannot be first-order types if t and t' are ordered types, they must be at the same type order	Irreflexive, antissymetric, transitive

Fig. 8. ML2 Editor verification of specialization constraints.

4.1. Addressing level blindness

The structure imposed on the instantiation relation and the notions of orderless and ordered types allow us to identify unsound multi-level structures. Some of these rules concern the stratified scheme imposed on ordered types. Some other rules apply to the relation between orderless and ordered types.

Concerning the instantiation relation: whenever instantiation involves solely ordered types, it is irreflexive, anti-symmetric and anti-transitive, leading to stratification rules, i.e. whenever an ordered type is instantiated its instances are at one order lower. Thus, ordered types cannot have orderless types as instances. However, when involving orderless types, there are situations in which instantiation can be reflexive (e.g., Type is instance of itself), symmetric (e.g., Entity is instance of Type and vice-versa) or transitive (e.g., OrderedType is instance of Type which is instance of Entity and OrderedType is also instance of Entity). Orderless types may have ordered types, orderless types and individuals as instances. All these MLT* rules are expressed as syntactic constraints in ML2, and follow assertions that were verified in Alloy.[6]

Table 1 summarizes the logical properties and rules for types involved in MLT* structural relations.

Concerning specialization, if the supertype is an ordered type, the subtype must be an ordered type at the same order (afterall, instances of the subtype are instances of the supertype, and if the instances of the supertype are all in the same order, so are the instances of the subtype). In case the supertype is an orderless type, the subtype may be also an orderless type or may apply only to instances at a specific order, being thus an ordered type (in which case the subtype selects from the instances of the supertype entities in the same order). Detection of violations of the rules involving specialization are shown in Fig. 8. First, there is a violation of the same order constraint in the specialization of Species (a second-order class) into Animal (a first-order class). Second, there is violation of irreflexive specialization for Thing (ML2 represents proper specializations). Notice, however, that specializations of orderless classes into ordered ones (Thing into Species and Animal) are allowed and are consistently represented. (Many other examples involving instantiation and specialization are discussed in Section 5.)

Concerning subordination, since it implies specializations between the instances of the involved types, whenever involving two ordered types, subordination can only hold between higher-order types of equal order. In cases the subordinate type is an ordered type, the other involved type may be an orderless type. Nevertheless, orderless types can only be subordinate to other orderless types. If orderless types were allowed to be subordinate to ordered types, as a consequence we would have orderless types specializing

---

Fig. 9. ML2 Editor verification of subordination constraints.

Fig. 10. ML2 Editor verification of powertype constraints.

ordered types, which leads to a logical contradiction, and is thus ruled out by MLT*. The example in Fig. 9 presents two constraint violations in a model involving the subordination of instances of Breed to instances of Species: the inconsistent subordination of a second-order class, SpeciesType, to a class of different order, Species; and an instance of Breed, Collie, that lacks specialization to an instance of Species (which is required by the subordination declaration in line 6).

4.2. More power to the powertype pattern

Some theorems arise from the MLT* definitions of is powertype of (a10) and categorization (a11) relations. These theorems also suggest concrete syntactic constraints to ML2. In addition to verification in Alloy, they have been subject to automated proof[7] with a specification written in the TPTP first-order logics syntax [26].

According to the is powertype of relation definition (a10), all instances of the powertype are specializations of the base type and, conversely, all specializations of the base type are instances of the power type. From this definition and from the notion of type equality, it follows that each type has at most one *powertype* (t1) and that each type *is powertype of*, at most, one other type (t2). These uniqueness constraints are syntactically verified in ML2 models, and the ML2 editor will flag as syntactically incorrect any model that includes more than one Cardelli powertype for a base type or that includes more than one base type for a Cardelli powertype.

t1 $\forall t_1, t_2(\text{isPowertypeOf}(t_1, t_2) \rightarrow \neg\exists t_3(t_1 \neq t_3 \land \text{isPowertypeOf}(t_3, t_2)))$

t2 $\forall t_1, t_2(\text{isPowertypeOf}(t_1, t_2) \rightarrow \neg\exists t_3(t_2 \neq t_3 \land \text{isPowertypeOf}(t_1, t_3)))$

Some other theorems entailed by MLT* concern the soundness of hierarchies of higher-order types. Theorem (t3) captures that if a type t_1 specializes a type t_2 then the (Cardelli) *powertype* of t_1 specializes the (Cardelli) *powertype* of t_2. Theorem (t4), in its turn, reflects that Odell powertypes (categorizations in MLT*) proper specialize the Cardelli powertype of a base type, i.e. if a type t_2 is the (Cardelli) *powertype* of t_1 and a type t_3 *categorizes* the same base type t_1 then all instances of t_3 are also instances of t_2 and, thus, t_3 proper specializes t_2. Finally, theorem (t5), captures a consequence of the partitions definition, namely: if two types t_1 and t_2 both *partition* the same type t_3 then it is not possible for t_1 to proper specialize t_2. Founded on these three theorems ML2 implements syntactic verification of higher-order types hierarchies on its models. Further, based on (t3) and (t4) ML2 checks the completeness of models, verifying if specializations between (Cardelli and Odell) powertypes are omitted.

t3 $\forall t_1, t_2, t_3, t_4(\text{specializes}(t_1, t_2) \land \text{isPowertypeOf}(t_3, t_1) \land \text{isPowertypeOf}(t_4, t_2)$
            $\rightarrow \text{specializes}(t_3, t_4))$

t4 $\forall t_1, t_2, t_3(\text{isPowertypeOf}(t_2, t_1) \land \text{categorizes}(t_3, t_1) \rightarrow \text{properSpecializes}(t_3, t_2))$

t5 $\forall t_1, t_2, t_3(\text{partitions}(t_1, t_3) \land \text{partitions}(t_2, t_3) \rightarrow \neg\text{properSpecializes}(t_1, t_2))$

ML2 also provides syntactic constraints founded on MLT* properties concerning the order of types involved in powertype relations. When involving ordered types, powertype relations (both Odell and Cardelli powertypes) only occur between types of adjacent levels. Further, a (Cardelli) powertype of an orderless type must also be an orderless type, while an Odell powertype

---

[7] The TPTP specification along with the proof reports for each of the theorems in this paper can be found in https://github.com/nemo-ufes/mlt-ontology.

Table 2
Summary of constraints and properties of MLT* relations between types.

Relation	Domain and Range	Constraint	Properties
isPowertypeOf(t, t')	orderless:orderless ordered:ordered	t cannot be a first-order type if t and t' are ordered types, t must be at a type order immediately above the order of t'	Irreflexive, antissymetric, antitransitive
categorizes(t, t') disjointlyCategorizes(t, t')	orderless:orderless orderless:ordered ordered:ordered	t cannot be a first-order type if t and t' are ordered types, t must be at a type order immediately above the order of t'	Irreflexive, antissymetric, nontransitive
completelyCategorizes(t, t') partitions(t, t')	orderless:orderless ordered:ordered	t cannot be a first-order type if t and t' are ordered types, t must be at a type order immediately above the order of t'	Irreflexive, antissymetric, antitransitive

(captured in MLT* by the notion of categorization) of an orderless type is not necessarily an orderless type. These latter constraints on the different notions of powertype relations implemented in ML2 are summarized in Table 2.

These constraints prevent the modeler from building unsound models, such as the one in Fig. 10 which includes an isPowertypeOf relation connecting a class declared as orderless to another declared as ordered, between TaxonomicRank and Taxon; and includes a completelyCategorizes relation involving two classes declared in the same order (second-order), between Taxon and Species. In this example, the issues arise from the mistaken declaration of TaxonomicRank as orderless (when it should be a third-order class), and the wrong usage of a categorization relation whereas Species should be connected to Taxon via specialization.

4.3. Specification and validation in alloy

In addition to model-checking the formal properties of the theory as discussed in the previous sections, Alloy's simulation and model-finding features can show that MLT* is a generalization of the two-level classification scheme. This is done by showing that adding an axiom imposing the existence of only one basic type and no orderless type still leads to a satisfiable theory. The same can be said for a three-level fixed architecture. See the formal specification for corresponding simulation directives.[8]

The Alloy specification also serves as a basis to verify and simulate ML2 models. A transformation from ML2 to Alloy described in [27] and implemented in the ML2 editor[9] produces Alloy code that imports the MLT* Alloy module. This guarantees that any verification and simulation of ML2 models adhere to the formal foundations.

ML2 model simulation provides an opportunity for modelers to investigate the consequences of their modeling choices, and thereby gain confidence on the domain adequacy of models. In particular, model simulation may reveal models that, while syntactically correct according to ML2, fail to capture the intended domain conceptualization. This case is illustrated in the sequel. Listing 4 and Fig. 11 present, respectively, an ML2 model on the domain of animal species and its simulation in the Alloy analyzer.

```
1   module species {
2     order 3 class TaxonomicRank;
3     order 2 class AnimalSpecies : TaxonomicRank
4       categorizes Animal
5     {
6       regularity warmbloodedSpecies : Boolean
7         determinesValue isWarmblooded
8       ref taxonAuthor : Person
9     };
10    class Person {
11      name : String
12    };
```

---

[8] The complete specification can be found in https://github.com/nemo-ufes/mlt-ontology.

[9] The Alloy transformation can be found in https://github.com/nemo-ufes/ml2-to-alloy
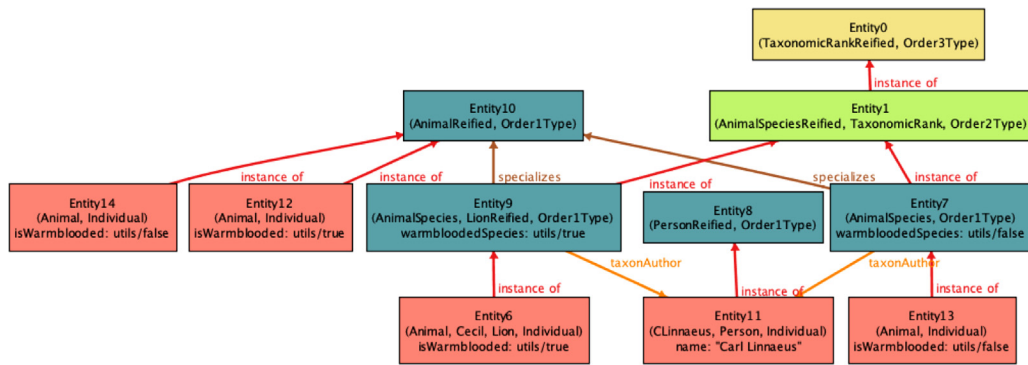
Fig. 11. Alloy simulation of a ML2 model on animal species.

```
13    class Animal {
14      isWarmblooded : Boolean
15    };
16    class Lion : AnimalSpecies specializes Animal {
17      warmbloodedSpecies = true
18      ref taxonAuthor = CLinnaeus
19    };
20    individual CLinnaeus : Person {
21      name = 'Carl Linnaeus'
22    };
23    individual Cecil : Lion;
24  }
```

Listing 4: ML2 model on animal species.

The simulation shows that – given Listing 4 – it is possible for instances of Animal (Entity14 and Entity12) *not* to instantiate an instance of AnimalSpecies. This is a consequence of the use of categorizes between AnimalSpecies and Animal. If this fails to capture the domain conceptualization, which typically requires all animals to have a species, the modeler could amend the model to use a complete categorization or a partitioning relation. Simulations of the amended ML2 model would then never produce such a scenario, as it would be ruled out intentionally.

In conclusion, beyond formal compliance with MLT* (which rules out a number of frequently occurring problems in practice as demonstrated in the following section), the ML2 editor facilitates a domain validation task.

5. Empirical assessment

Wikidata [28] is a large, free and open knowledge base that is part of a family of projects surrounding the Wikipedia project. Its aim is to provide a structured representation of Wikipedia's content. All data representation in Wikidata is based on triples (i.e., subject–predicate–object statements), often serialized into RDF syntax. As a result, Wikidata can be conceived of as a large knowledge graph, constantly curated by humans and software agents. Nodes of this graph are called "items". Items in Wikidata can be used to represent instances and classes alike. For example, there are items to represent "London" as well as the "city" class. Edges of the Wikidata graph are formed by applying "properties". The instance of property (wd:P31), which is Wikidata's equivalent of RDF's rdf:type, serves to connect a node representing an instance to a node representing a class it instantiates. For example, the item representing "London" is related to the item representing the class "city" through the instance of property, to represent the fact that London is a city. Chains of instantiation are simply chains in the graph using instance of several times. The native Wikidata approach can be considered level-blind, i.e., with no built-in mechanisms to apply level segregation and/or cohesion principles [29].

An empirical analysis performed in 2015 [11] indicated at that time that Wikidata presented thousands of concepts involved in multi-level anti-patterns. Thousands of entities were found to be involved in violation of MLT level stratification rules. Since then, however, Wikidata has incorporated some support for level stratification relying on a basic scheme from OpenCyc [19] similar to the one presented in Fig. 5. In light of these advances, we revisit here an assessment of the representation of multi-level domains in Wikidata.[10] Despite the advances, we show that there are still many problematic multi-level structures in Wikidata. We show that ML2 can identify such problems as syntactically invalid expressions. This provides empirical evidence of the value of the language in uncovering representation problems in a realistic setting.

---

[10] This new empirical analysis was finalized during second semester of 2020 and may be affected by updates to Wikidata's dataset. Please refer to Wikidata dumps generated within the same time window when verifying the results of this research.

5.1. Wikidata's approach to multi-level modeling

Wikidata's current approach to multi-level modeling relies on OpenCyc's basic scheme [19], which is nearly equivalent to MLT*'s basic scheme of Fig. 5, including both orderless and ordered entities. As a result of employing this scheme, Wikidata can represent both entities that follow a rigid stratification principle as well as those that do not, using instantiation and specialization (P279 termed subclass of) to the entities in the basic scheme to identify an entity's order. The following items compose this basic scheme:[11]

- something (Q35120): a class whose instances are all entities; equivalent to MLT*'s Entity;
- (meta)class (Q23960977): a subclass of something whose instances are all classes; equivalent to MLT*'s type;
- individual (Q23958946): a subclass of something whose instances are all individuals; equivalent to MLT*'s Individual;
- fixed-order metaclass (Q23959932): a subclass of (meta)class whose instances are classes that fall into a strict stratification scheme; equivalent to MLT*'s Ordered Type;
- variable-order metaclass (Q23958852): a subclass of (meta)class whose instances are classes that do not fall into a specific level; equivalent to MLT*'s Orderless Type;
- first-order metaclass (Q24017414): a subclass of fixed-order metaclass whose instances are second-order classes; equivalent to MLT*'s Second-Order Type;
- second-order metaclass (Q24017465): subclass of fixed-order metaclass whose instances are third-order classes; equivalent to MLT*'s Third-Order Type;
- third-order metaclass (Q24027474): subclass of fixed-order metaclass whose instances are fourth-order classes;

In addition to the list above, Wikidata also defines the concept class (Q16889133), which seems to be (at least from most common usage in current Wikidata) equivalent to MLT*'s First-Order Type. We leave this class outside our analysis, as this entity is defined outside Wikidata's basic scheme outlined above, which may be a consequence of an ongoing effort of reconciling legacy data.

5.2. ML2-based analysis of Wikidata

In this sub-section, we consider three anti-patterns of multi-level representation that we identified in Wikidata. The number of occurrences of these anti-patterns show that, even with the classification of items with the multi-level basic scheme, errors in multi-level taxonomies still remain common in Wikidata.

For each of the anti-patterns, we show unsound fragments of the Wikidata graph. We also show that, when these fragments are formalized as corresponding ML2 models, the problems become plain syntactic errors in the language.

5.2.1. Anti-pattern 1: Individuals with instances

The first anti-pattern concerns items classified as individuals that themselves have instances. These items can be identified with the SPARQL query in Listing 5. It selects entities ?x in Wikidata that instantiate individual (Q23958946) (directly or through its subclasses), and, at the same time, themselves have instances (?y):

```
1  PREFIX wd: <http://www.wikidata.org/entity/>
2  PREFIX wdt: <http://www.wikidata.org/prop/direct/>
3  SELECT DISTINCT ?x ?y WHERE {
4    ?x (wdt:P31/(wdt:P279+)) wd:Q23958946.
5    ?y wdt:P31 ?x
6  }
7  LIMIT 45000 # results in 58072 ms
```

Listing 5: Instances of Individual that have instances of their own.

This SPARQL query selects pairs in Wikidata that violate (t6), which is a direct consequence of the definition of individual (a2).

t6 $\neg \exists x, y(\mathsf{iof}(x, \mathsf{Individual}) \land \mathsf{iof}(y, x))$

When applied to Wikidata's SPARQL endpoint, this query reaches the established limit of 45,000 matches (this limit was established to avoid the one minute timeout in Wikidata's server). In other words, there are at least 45,000 items in Wikidata's knowledge graph classified as individuals that themselves have instances!

These problems fall into two main cases: (i) ?x is indeed supposed to represent an individual but is inadequately instantiated by some other entity ?y (and thus the error should be rectified by fixing the statements that instantiate ?x), or (ii) ?x is inadequately classified as an individual, when it in fact represents a class (and hence the error should be rectified by reclassifying ?x).

An example for the first case involves the United Nations (Q1065) item, which clearly stands for the multilateral organization. Inadequately declared "instances" of the United Nations concept include Green Climate Fund (Q3075923), United Nations Department of Global Communications (Q3708827) and Universal Postal Union (Q17495). Even though these items are most

---

[11] Note that there are some conflicting statements and usage problems in Wikidata, so this is our best interpretation of the intended semantics.

```
 3        class Individual;
 4
 5        individual UnitedNations : Individual;
 6
 7        individual GreenClimateFund : UnitedNations;
 8        individual UnitedNationsDepartmentofGlobalCommunications : UnitedNations;
 9        individual UniversalPostalUnion : UnitedNations;
```

Fig. 12. Representation of Wikidata's United Nations (Q1065) example.

```
10        class Individual;
11        class Agent specializes Individual;
12        class Organization specializes Agent;
13        class ReligiousOrganization specializes Organization;
14        class ReligiousAdministrativeTerritorialEntity specializes ReligiousOrganization;
15
16        individual DioceseOfTheCatholicChurch : ReligiousAdministrativeTerritorialEntity;
17        individual RomanCatholicDioceseOfOslo : DioceseOfTheCatholicChurch;
18        individual RomanCatholicDioceseOfAuckland : DioceseOfTheCatholicChurch;
```

Fig. 13. Representation of Wikidata's diocese of the Catholic Church (Q3146899) example.

likely somehow related to the United Nations, the relation between them is not instantiation, as there are no repeatable properties defined by the later that could be exemplified by the former three, which is a defining characteristic of instantiation.

A noticeable example of the second case (when an item in incorrectly classified as an individual) is the case of the item diocese of the Catholic Church (Q3146899), described as "a diocese of the Catholic Church under the supervision of a bishop". This description contributes to the interpretation of the item not as a singular individual, but rather as a whole category of individual dioceses (religious administrative territorial entities) that have these characteristics. This item has over 2,000 instances in Wikidata, with items for the various individual dioceses of the Catholic Church worldwide, such as the Roman Catholic Diocese of Oslo (Q44171) and the Roman Catholic Diocese of Auckland (Q369407). These two aspects contribute to the conclusion that the concept was improperly classified as an individual rather than a first-order type. Given the apparently obvious semantic problem, how did this escape the various curators of Wikidata? We explore this issue further in the sequel, and conclude that automated support would prevent such errors from arising.

The classification of diocese of the Catholic Church (Q3146899) as an individual (Q23958946) is the consequence of a long chain of instantiation/specialization relations, summarized in Listing 6:

```
 1    # "diocese of the Catholic Church" instantiates
 2    # "religious administrative territorial entity":
 3    wd:Q3146899 wdt:P31 wd:Q20926517.
 4    # "religious administrative territorial entity" specializes
 5    # "religious organization":
 6    wd:Q20926517 wdt:P279 wd:Q1530022.
 7    # "religious organization" specializes "organization":
 8    wd:Q1530022 wdt:P279 wd:Q43229.
 9    # "organization" specializes "agent":
10    wd:Q43229 wdt:P279 wd:Q24229398.
11    # "agent" specializes "individual":
12    wd:Q24229398 wdt:P279 wd:Q23958946.
```

Listing 6: Classification chain connecting Q3146899 to individual.

Because religious administrative territorial entity specializes individual indirectly, all its instances should be individuals, including the item we are considering in this analysis (diocese of the Catholic Church). We can only speculate on how this situation came to be. Perhaps, the agents that provided information regarding Q3146899 to Wikidata indeed intended to represent the concept as an individual. However, its textual description and the presence of further specialization relations towards diocese (Q665487) and religious administrative entity (Q51041800) suggest that this is not the case (afterall it is only classes and not individuals that are related by specialization). Most likely, the deep chain of relations presented in Listing 6 obscures the relation from the item to individual, and in fact, specialization and not instantiation should relate diocese of the Catholic Church to diocese (Q665487) and religious administrative entity, making the item a first-order class, which is consistent with the so many instantiations.

The representation of these examples in ML2 cannot be parsed as, according to the language's metamodel, only classes can be the target of instantiation relation. This is presented in Figs. 12 and 13, where red underlining indicates to the user references that could not be resolved by the parser.

5.2.2. Anti-pattern 2: Entity instantiates entity in the same order

In the second anti-pattern, we investigate the presence of strictly stratified types of the same order connected through instance of relations. More specifically, we apply the query for this anti-pattern in Listing 7 selecting ?x and ?y such that both are instances

```
21      order 3 class FirstOrderMetaclass;
22      order 3 class Rank_Q4189293 specializes FirstOrderMetaclass;
23      order 3 class Rank_Q486983 specializes Rank_Q4189293;
24      order 3 class AcademicRank specializes Rank_Q486983;
25
26      order 2 class Profession : FirstOrderMetaclass;
27
28      class ResearchProfessor : Profession, AcademicRank;
29
30
31
32
33
```

❌ Invalid instantiation of AcademicRank

Fig. 14. Representation of Wikidata's research professor (Q27177003) example.

of first-order metaclass (Q24017414) (directly or through specializations) and ?x instantiates ?y. The anti-pattern is a violation of stratification for ordered types.

```
1   PREFIX wd: <http://www.wikidata.org/entity/>
2   PREFIX wdt: <http://www.wikidata.org/prop/direct/>
3   SELECT DISTINCT ?x ?y WHERE {
4     ?x wdt:P31/(wdt:P279*) wd:Q24017414.
5     ?y wdt:P31/(wdt:P279*) wd:Q24017414.
6     ?x wdt:P31/(wdt:P279*) ?y.
7   }
```

Listing 7: Same order entities holding an instance-of relation.

The query selects pairs in Wikidata that violate (t7), which is a consequence of instantiation only applying to entities in adjacent orders.

t7 $\neg\exists x, y(\text{iof}(x, 2\text{ndOT}) \land \text{iof}(y, 2\text{ndOT}) \land \text{iof}(x, y))$

Over 4900 second-order types are found in this query, a significant phenomenon considering that Wikidata contains about 13,000 second-order types currently (37,7%). In terms of domains related to this constraint violation, most seem to be related to instances of taxonomies of social roles, including profession (Q28640), military rank, (Q56019), and baronetcy (Q18759100), the latter an example of aristocratic rank. One of such violations is presented in Listing 8 for the case of research professor (Q27177003).

```
1   # "research professor" instantiates "profession":
2   wd:Q27177003 wdt:P31 wd:Q28640.
3   # "profession" instantiates "first-order metaclass":
4   wd:Q28640 wdt:P31 wd:Q24017414.
5
6   # "research professor" instantiates "academic rank":
7   wd:Q27177003 wdt:P31 wd:Q486983.
8   # "academic rank" specializes "rank" (social role)
9   wd:Q486983 wdt:P279 wd:Q4189293.
10  # "rank" (social role) specializes "rank" (hierarchical)
11  wd:Q4189293 wdt:P279 wd:Q4120621.
12  # "rank" (hierarchical) specializes "first-order metaclass"
13  wd:Q4120621 wdt:P279 wd:Q24017414.
```

Listing 8: Classification chain connecting Q27177003 to first-order metaclass.

The research professor class has two chain of classification relations connecting it to first-order metaclass. The first chain is an instance-of relation towards profession, which in turn holds an instance-of relation towards first-order metaclass, resulting in an interpretation of profession as a second-order class and research professor as a first-order class. This interpretation is in line with the description of research professor, whose instances are individual professors (e.g., Marie Currie or Albert Einstein). The second chain is an instance-of relation towards academic rank, which in turn holds a chain of subclass of relations towards rank (Q4189293), rank (Q4120621), and finally first-order metaclass, resulting in an interpretation of academic rank, rank (Q4189293), and rank (Q4120621) as a third-order classes and research professor as a second-order class. The representation of this model excerpt in ML2 (Fig. 14) highlights this inconsistent classification that places a concept simultaneously in two distinct levels.

In order to solve this violation, we must understand which interpretation is the correct one, allowing us to reflect where lies the problem. As ML2 requires explicit declaration of an entity's order, we have opted in Fig. 14 to follow the first interpretation

```
21   order 3 class FirstOrderMetaclass;
22   order 3 class Rank_Q4189293 specializes FirstOrderMetaclass;
23   order 3 class Rank_Q486983 specializes Rank_Q4189293;
⊗24  order 3 class MilitaryRank : FirstOrderMetaclass specializes Rank_Q486983;
25
26                                    ⊗ Invalid instantiation of FirstOrderMetaclass
27
28
```
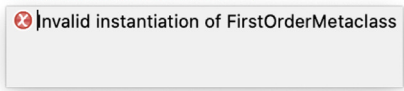
Fig. 15. Representation of Wikidata's military rank (Q56019) example.

of research professor as a first-order class as its description suggests. Therefore we conclude that either academic rank does not classify research professor, or it should be a second-order class, instantiating first-order metaclass rather than specializing it. Other examples of ranks in Wikidata, including taxonomic rank (Q427626) which is aligned with our earlier example discussed earlier in Fig. 1 suggest that the former is more likely, leading us to believe that academic rank is indeed a third-order class that should not classify research professor, but other second-order classes.

5.2.3. Anti-pattern 3: Explicitly inconsistent order declarations

The third query we explore selects items defined *simultaneously* as instances of first-order metaclass (i.e., items that represent second-order types) and as specializations of first-order metaclass (i.e., items that represent third-order types). The query for this anti-pattern is presented in Listing 9.

```
1   PREFIX wd: <http://www.wikidata.org/entity/>
2   PREFIX wdt: <http://www.wikidata.org/prop/direct/>
3   # Query for ?x such that ?x instantiates and specializes Q24017414
4   # (first-order metaclass)
5   # 1266 results in 1731 ms
6   SELECT DISTINCT ?x WHERE {
7     ?x wdt:P31/(wdt:P279*) wd:Q24017414.
8     ?x (wdt:P279+) wd:Q24017414.
9   }
```

Listing 9: Instantiation and specialization of basic type.

The query selects pairs in Wikidata that violate (t8), which is a consequence of instantiation only applying to ordered types in adjacent orders, and specialization only applying to ordered types in the same order.

t8 $\neg\exists x(\mathrm{iof}(x, \mathrm{2ndOT}) \wedge \mathrm{specializes}(x, \mathrm{2ndOT}))$

This query results in 1266 entities, out of the 13,260 entities that are instances of first-order metaclass (9,5%). Among the entities involved in anti-pattern 3, a number of them represent social roles which are also involved in anti-pattern 2 due to the same indirect relations. For example, the class military rank (Q56019) which both instantiates and specializes first-order metaclass, as detailed in Listing 10 and highlighted in Fig. 15.

```
1   # "military rank" instantiates "first-order metaclass"
2   wd:Q56019 wdt:P31 wd:Q24017414.
3
4   # "military rank" specializes "rank" (social role)
5   wd:Q56019 wdt:P279 wd:Q4189293.
6   # "rank" (social role) specializes "rank" (hierarchical)
7   wd:Q4189293 wdt:P279 wd:Q4120621.
8   # "rank" (hierarchical) specializes "first-order metaclass"
9   wd:Q4120621 wdt:P279 wd:Q24017414.
```

Listing 10: Classification chain connecting Q524980 to first-order metaclass.

In order to assess the order of an entity, we must identify a chain of classification relations (i.e., instantiations and specializations) from the entity to Wikidata's basic scheme. For example, an entity specializing first-order metaclass is in the same level of it (i.e., a third-order class) whereas an entity instantiating it is in one level below (i.e., a second-order class). As detailed, the chains connecting military rank to first-order metaclass do not assessing whether it is a second-order or third-order class as they are connected through both instantiation and specialization. Still, following the same reasoning of academic rank, it appears to be the case that military rank should represent a third-order class.

6. Related work

In this section, we review other efforts most closely related to our work. We establish key requirements for a multi-level conceptual modeling language, substantiating these requirements with sources from the literature on multi-level modeling and justifying them based on intended usage scenarios (i.e., representation needs) (based on [30]). We then examine the capabilities of the various multi-level approaches using the established features.

6.1. Features for multi-level conceptual modeling approaches

First of all, given the nature of a multi-level scheme, an essential feature for a multi-level modeling approach is the ability to *represent entities of multiple (related) classification levels*, capturing chains of instantiation between the involved entities (Feature **F1**). In opposition to the traditional two-level scheme, this feature allows multi-level approaches to classify classes that break with the class/individual divide.

The size of the instantiation chains may vary according to the nature of the represented phenomena and the model's purposes. Because of this, a general-purpose multi-level modeling language should *allow the representation of an arbitrary number of classification levels* [6,11,31] (Feature **F2**) (including the two-level scheme as a special case).

Another feature of a multi-level modeling approach is to *define guiding principles for the organization of entities into levels* (Feature **F3**). These principles should guide the modeler on the adequate use of classification (instantiation) relations. This feature is widely adopted in multi-level modeling, being present, e.g., in ML2, through the guiding notion of "order"; in Wikidata, with its basic scheme; and in techniques adhering to the strict metamodeling principle [32].

While these principles are intended to guide the modeler in producing sound models, they should not obstruct the representation of genuine multi-level phenomena. The strict metamodeling principle, for example, excludes from the domain of inquiry abstract notions such as a universal Type or, the even more abstract notion of Thing. We conclude that to *support the representation of types that defy a strictly stratified classification scheme* (Feature **F4**) is an important feature to deal concepts such as Resource, Property, or Thing, present in scenarios like the Semantic Web [20], ontology engineering [14,19,33], and others.

Structural relations are often employed to support multi-level approaches, as discussed here with the notions of Odell [18] and Cardelli [17] for powertypes. Being employed in a number of modeling approaches, among which the Unified Modeling Language (UML) [34], we consider the ability to *represent rules that govern the instantiation of related types at different levels* (Feature **F5**) among the listed features.

In addition to the features supporting the representation of instantiation relations, we also consider features regarding how properties of classes and individuals reflect the particularities of multi-level modeling in a specific approach. Since these approaches are characterized for admitting the presence of types as instances of other types, we consider the feature of supporting the *representation of properties (attributes and relationships) of types* as well as the assignment of values to their instances (Feature **F6**).

While in the traditional two-level scheme, types capture invariant properties of individuals, in higher-order types invariant properties may also constrain properties at lower levels. This feature is frequently present in approaches following the strict metamodeling principle [22] and is present in ML2 in the form of *regularity features*. We consider among the list of multi-level features the ability to *support rules relating properties of entities in different levels* (Feature **F7**).

Finally, in various domains, there are relations that may occur between entities of different classification levels [35]. For example, in Wikidata, the property taxon author (P405) relates a first-order class, such as Lion (Q140), to an individual, which in this case is Carl Linnaeus (Q1043). Thus, a multi-level modeling language should allow the *representation of domain relations between entities of various classification levels* (Feature **F8**).

6.2. Capabilities of current multi-level modeling approaches

In this section, we position ML2 with respect to the existing work in multi-level representation approaches regarding the list of features from Section 6.1. We consider the following (multi-level) modeling approaches: UML [34], DeepTelos [7], DeepJava [36], Melanee [6], M-Objects [5], MetaDepth [8], Kernel [37], OMLM [38], SLICER [39], TOTEM [40], and MultiEcore [41]. Table 3 summarizes our evaluation of the various modeling approaches: a plus sign (+) indicates support for the feature, a minus sign (−) indicates no support, while plus/minus (±) indicates partial support. This also serves as a comparison to ML2, which implements all of the listed features.

In the UML 2.5.1 specification [34], a class plays the role of *powertype* whenever it is connected to a generalization set composed by the generalizations that occur between a base classifier and the instances of the powertype. Given that generalization sets only exist when specializations of the base type are modeled, the UML cannot capture simple multi-level models in which instances of a powertype are omitted. As discussed in [12], this rules out simple models such as DogBreed categorizing Dog, when specific breeds are omitted. Hence, we consider the UML to only partially support **F1**. In UML, chains of instantiation of arbitrary size can be captured by cascading the powertype pattern iteratively (again requiring the use of explicit specializations in generalization sets), thus partially supporting **F2**. Further, the UML specification does not provide principles to guide the organization of entities into (classification) levels. The only rule in UML concerning the consistency of instantiation chains aims at avoiding a "powertype" to be an instance of itself. Due to this incompleteness, it does not support **F3**. This very same constraint rules out some orderless types, such as the type Type. Therefore, we consider that the UML only partially supports **F4**. We consider that the notion of powertype in UML corresponds to MLT's notion of categorization, failing to capture Cardelli's powertype, since all instances of the powertype

Table 3
Summary o multi-level approaches and their capabilities according to the list of multi-level features.

Features	UML	DeepTelos	DeepJava	Melanee	M-objects	MetaDepth	Kernel	OMLM	SLICER	TOTEM	MultiEcore
F1: represents entities of multiple classification levels	±	+	+	+	+	+	+	+	+	+	+
F2: arbitrary number of classification levels	±	+	+	+	+	+	+	+	+	+	+
F3: defines guiding principles for organization of models	−	±	+	+	+	+	−	+	±	+	+
F4: types that defy a stratified classification scheme	±	+	−	±	−	±	+	−	−	±	±
F5: represent rules to govern instantiation of related types	±	±	±	−	−	−	−	−	+	−	−
F6: represents features and feature assignments	−	+	+	+	+	+	+	+	+	+	+
F7: relates features of entities in different levels	−	−	±	+	−	+	−	−	+	+	+
F8: domain relations between entities in various levels	−	+	+	−	+	+	+	−	+	+	−

must be members of an identified generalization set. Thus, we consider that **F5** is only partially supported by UML. Given that instances of powertypes cannot have values assigned to their features, UML does not support **F6**, **F7** and **F8**.

DeepTelos is a knowledge representation language that approaches multi-level modeling with the application of the notion of "most general instance (MGI)" [7]. In [42], the authors revisit the axiomatization of Telos and add the notion of MGI to Telos' formal principles for instantiation, specialization, object naming and attribute definition. The notion of MGI can be seen as the opposite of Odell's powertype relation. For example, to capture that "Tree Species" is a "powertype" (in Odell's sense) of "Tree", in DeepTelos it would be stated that "Tree" is the "most general instance" (MGI) of "Tree Species". Considering that the MGI construct allows representing entities in multiple classification levels and that DeepTelos allows representing chains of MGI to represent as many levels as necessary, we consider that DeepTelos supports features **F1** and **F2**. DeepTelos builds up on Telos, whose architecture defines the notions of simple class and w-class, which are analogous to the notions of ordered and orderless types we use. Nevertheless, stratification rules for simple classes (constraining specialization and cross-level relations) are not provided. Thus, we consider that it partially supports **F3** and that it supports **F4** with the notion of w-class. Considering that DeepTelos provides only the concept of MGI to constrain the instantiation of types in different levels, not elaborating on the nuances of the relations between higher-order types and base types, we consider that it partially supports **F5**. It supports both the declaration and assignments of attributes (**F6**), but it does not include constraints over related attributes at different levels as one of its primitives (**F7**). Finally, DeepTelos also allows users to declare relations between entities in different classification levels, thus supporting **F8**. See [43] for an in-depth analysis of the relation between DeepTelos and MLT*, in which some of the missing features have been incorporated into a DeepTelos-based MLT* deductive realization.

DeepJava is an extended version of Java that supports multi-level mechanisms for programming languages [36]. The language allows the specification of potencies for Java classes and fields along with instantiation for classes. In DeepJava, the potency of an element denotes the maximum depth of its instantiation chain, or how many times a type can be instantiated. Through this mechanism, DeepJava is able to define entities at an arbitrary number of classification levels, defining the level on which each entity sits, thus supporting **F1**–**F3**. As the language only accounts for defined potencies with direct instantiation, it does not account for entities that defy the stratification in levels, not supporting **F4**. Applying potencies in tandem with specializations, DeepJava allows representing that all instances of a (higher-order) type specialize another type. Considering that such mechanism maps Odell's

notion of powertype (and MLT's notion of characterization), not elaborating on further nuances of the relations between higher-order types and base types, we consider that it partially supports **F5**. As a programming language, DeepJava supports references between any objects in memory, and both feature specification and assignment at any classification level (except at the highest one, since a pure Java class does not have features to which values can be assigned). 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. Therefore, DeepJava supports **F6** and **F8**, and partially supports **F7**.

Melanee [6] is a tool that supports multi-level modeling founded on the notions of strict metamodeling, clabjects and potency. It is based on the idea of defining clabjects and fields (attributes and slots) within the levels of a strictly stratified scheme (i.e., strict metamodeling [32]) and assigning to both clabjects and fields a potency, which defines how deep the instantiation chain produced by that clabject or field can become. This allows Melanee to represent entities in multiple classification levels (**F1**), organizing and capturing the instantiation chains allowing an arbitrary number of levels (**F2**), and providing users guiding principles for the organization of models (**F3**). Melanee also defines "star potency" as a means to support the representation of types having instances of different potencies. While this allows the representation of types that defy a stratified scheme (**F4**), star potency does not allow self-instantiation, which is required for the abstract types we have dealt with here. Therefore, we consider that Melanee partially supports **F4**. In Melanee, no constructs are provided to capture rules concerning instantiation of related types at different levels (**F5**). For example, it is not possible to represent in Melanee that "CellphoneModel" categorizes (in MLT* sense) "Cellphone", and thus, it is not able to capture that every instance of "CellphoneModel" must specialize "Cellphone". Further, in Melanee, instantiations are the only relations that may cross level boundaries and, thus, it is unable to capture certain domain scenarios in which an entity is related to other entities at different instantiation levels (**F6**). For example, consider a scenario in which every instance of "CellphoneModel" has a "designer" being an instance of "Person" and every instance of an instance of "CellphoneModel" (i.e., every instance of "Cellphone") has an "owner" which is also an instance of "Person". Since domain relations in Melanee cannot cross levels, both "Person" and "Cellphone" must be placed in the same level to capture the "owner" relation. Because its instances are specializations of "Cellphone", "CellphoneModel" must be placed in one level higher. This makes it impossible to capture the "designer" relation, as it would cross level boundaries (which, once more, is not allowed in Melanee). Concerning domain features, Melanee supports both the representation of features of types as well as the attribution of values to those features (**F7**). Finally, the combination of the notions of attribute durability and mutability [9] allows one to relate features of entities in different levels (**F8**). For example, it allows one to capture that instances of "CellphoneModel" prescribe the exact screen size their instances must have. Note that it supports directly only one of the six types of regularity features covered in ML2 (namely, the one in which the value is fully determined).

In [5], the authors propose a multi-level modeling approach founded on the notion of m-object. M-objects encapsulate different levels of abstraction that relate to a single domain concept, and an m-object can concretize another m-object. The concretize relationship comprises indistinctive classification, generalization and aggregation relations between the levels of an m-object [5]. This approach allows the representation of entities in an arbitrary number of levels relating them through chains of concretize relationships, we consider that it supports **F1** and **F2**. Given that the approach adopts a stratified schema in which concretize relationships may only relate types at adjacent levels, we consider that it supports **F3** and does not support **F4**. Further, since the concretize relationships are the only structural relationships that cross level boundaries, the approach fails to support **F5**. In [35], the authors observe that the approach was unable to capture certain scenarios in which there are domain relations between m-objects at different instantiation levels. To address this limitation, the approach was extended with the concept of Dual-Deep Instantiation, which allows the representation of relations between m-objects at different instantiation levels through the assignment of a potency to each association end, thereby supporting **F6**. Finally, it provides support to represent features of types (**F7**), but it does not include support to explain the relationship between attributes of entities in different classification levels (not supporting **F8**).

MetaDepth is a textual multi-level modeling language founded on the same notions of clabject, potency, durability and star potency used by Melanee. Differently from Melanee, MetaDepth supports the representation of domain relationships as references, such that each reference has its own potency (a solution similar to the one adopted in Dual-Deep Instantiation [35]), allowing the representation of domain relations between clabjects at different instantiation levels. Therefore, MetaDepth supports all the features Melanee supports, and further supports **F6**.

Kernel [37] was proposed as a foundation for model-based language engineering. A Kernel class is also an object and, as such, it can instantiate other classes iteratively, thereby supporting **F1** and **F2**. It supports **F4**, **F6** and **F8**, since it is rather unconstrained in order to support the definition of various multi-level modeling mechanisms. Given its focus as an agnostic basis, it does not aim at directly supporting organization principles, structural rules nor deep instantiation mechanisms (therefore it does not aim at supporting **F3**, **F5** or **F7**). Nonetheless, this focus of Kernel allows it to describe others approaches, such as potency-based and powertype-based approaches.

The Open integrated framework for Multi-Level Modeling (OMLM) [38] is a multi-level approach focused on a strict separation of concerns between three dimensions: the ontological dimension, concerned with the subject domain; the linguistic dimension, concerned with the linguistic elements involved in the representation of the domain; and the realization dimension, which focus on mapping models to a implementation target of choice. By making use of Flora-2 [44], an F-Logic dialect, OMLM supports a clabject-based representation of multi-level domains, with the advantage of allowing the user to extend the language by adding constructs and syntactic rules. Originally, OMLM supports the representation of entities in multiple (unbound and related) classification levels, supporting **F1**–**F3**. OMLM, however, in its ontological dimension, does not support types that defy the organization of entities into levels (**F4**), solely allowing instantiation relations between adjacent levels. The language also does not support **F5**, as there are no other relations besides instantiation for guiding the classification of entities. Attributes in OMLM are considered single-potency

elements, i.e., elements that can be instantiated only once in the ontological dimension. This treatment of attributes is able to support **F6**, but not **F7**, since there is no mechanism for supporting the representation of related features at different levels. In a previous version of OMLM called MiF [45], the same authors claim that their language does not support cross-level domain relations (**F8**), even though can potentially be extended in that sense.

Selway et al. [39] propose the SLICER conceptual framework, which also accounts for multi-level models. SLICER provides to the user a set of level-aware relations that enable multi-level modeling, such as specializations, instantiations and powertype ("subset by specification") relations. In SLICER, not only instantiation characterizes the transition between levels, but also specialization when properties are added to a super type. Some rules for levels are provided using these relations. SLICER is able to support **F1** and **F2** through the definition of entities in an unbound number of classification levels, but we consider it to partially support **F3** since the rules for organization into levels are rather loose (despite being well defined). The rules imposed on specialization and instantiation prevent some general types such as Type and Thing from being represented (e.g., because of self-instantiation), lacking support to **F4**. SLICER is able to support **F5** through the Subset-by-Specification relation, which has the same semantics of Odell's notion of powertype [18] and includes variations based on complete and disjoint constraints. Finally, the language supports both shallow and deep instantiation, and does not impose constraints over domain relations between entities of different levels (supporting **F6**–**F8**).

TOTEM [40] is a modeling approach consisting of a tool designed for the development of Ecore-based multi-level models. The goal in TOTEM is to enable the user to harness the available Ecore MDE technologies (e.g., executables as well as constraining and transformation languages). In order to achieve this goal, TOTEM models are built following the metamodeling principle and employing a potency-based approach, much like Melanee and MetaDepth. These models may be compiled then (i.e., transformed) into Ecore meta-models enriched with constraints to preserve their consistency in Ecore's two-level scheme. TOTEM follows MetaDepth's choices for the representation of multi-level models, having similar capabilities in terms of the list of features discussed here. Therefore, TOTEM support features **F1**–**F3**, and **F6**–**F8**, while partially supporting **F4** due to a limitation in regard to anti-symmetric instantiation (e.g., Thing as instance of Type), and not supporting **F5**. This analysis for supported features, however, considers TOTEM models prior to their compilation, as the user of a model that was compiled into Ecore must be aware of further representation limitations. For instance, in Ecore, TOTEM's instantiation relations are transformed into specializations of abstract classes and, for the perspective of other Ecore technologies, a TOTEM model capturing "Lion instantiates Species" would be processed as "Lion specializes Species".

Finally, MultiEcore [41,46] is a multi-level approach that enables multi-level modeling in Ecore. MultiEcore approach organizes models into levels (e.g., M1, M2, M3 and so on) and generates from each level both an Ecore metamodel and an instance model, capturing class and instance facets of its concepts, respectively. Bidirectional transformations are responsible for keeping a consistent representation between the different facets in a transparent manner to the user. Following the potency-based approach, MultiEcore supports features **F1**–**F3**, also having the aforementioned limited support of other potency-based approaches to **F4** and no support for **F5**. In regard to attributes and references, MultiEcore supports **F6** and **F8**, but the constrained usage of potency 1 to attributes prevents it from supporting **F7**.

Other authors also propose guidelines for the comparison of multi-level approaches, each having specific considerations, such as implementation aspects or intended application [40,47,48]. These proposals contribute to the effort of achieving a common set of characteristics that support the user in selecting an approach suitable for the application at hand.

When evaluated in light of the feature set of [40], for instance, ML2 is recognized for natively supporting a multi-level representation mechanism and deep instantiation [22]. However, ML2 does not support extension mechanisms that allow instances to include unforeseen properties, a mechanism often necessary in (flexible) language engineering. ML2 opts to require explicit property definition in classes.

7. Conclusion

In this paper, we have presented the ML2 multi-level conceptual modeling language. The language harnesses the conceptualization formalized in MLT*, reflecting the theory's definitions in its constructs and syntactical constraints. Rules incorporated in ML2 have been implemented in an Eclipse-based editor that supports the live verification of models to ensure adherence to the theory. The use of a formally-verified semantic foundation (employing both Alloy and first-order logics) is one of the distinctive features of ML2. In fact, the formal techniques were instrumental in the design of the theory and then language. Simulation of the theory has revealed the various multi-level structures that it supports, in a generalization of the two level scheme. Simulation has also revealed recurrent patterns that led to the formulation of theorems which in turn were model checked through assertions in Alloy and automated provers. In Alloy, these assertions are verified exhaustively by the Alloy Analyzer within a finite scope setting. The verified assertions formed the basis for the definition of semantically-motivated syntactic constraints for the language.

Over the decade, a number of languages have been analyzed for their adherence to reference theories *a posteriori*. This has been very fruitful to reveal a number of deficiencies in languages, under the banner of ontological analysis [49]. Rosemann, Green and Indulska argue that a language's abstract syntax should be isomorphic to their underlying ontology, i.e., there should be a one-to-one mapping between the constructs of a language and the concepts of its ontology. Analyzing this isomorphism reveals issues such as *construct overload* and *construct deficit* in a language. In this paper, we have shown that reference theories can have a prominent role early in a language's design cycle, preventing ontological deficiencies from appearing in the first place. The reference theory is used during language design instead of *a posteriori*.

The formalization also forms the basis for the simulation and verification of ML2 models, through model transformation from ML2 into Alloy. This means that multi-level models can profit from the same model simulation and verification available for two-level

models [50]. In our current work, we are extending ML2 and the corresponding transformation to support the specification of OCL constraints in ML2. This will allow us to specify invariants and derivation rules in ML2.

The language was designed to offer expressiveness to the modeler by addressing a comprehensive set of features for representing domains dealing with multiple levels of classification. In particular, we have provided support for the representation of types in classification schemes that transcend a rigid two-level structure. This means that ML2 is able to capture general types that defy a strictly-stratified scheme, but can still leverage the detection of errors for the so-called ordered types. We have shown that this support is key to identifying errors that are not perceived by users in realistic settings. In fact, we have shown that the inclusion of classes of a basic scheme for multi-level representation in Wikidata (following OpenCyc [19]) has not been enough to equip users with the means for obtaining correct multi-level taxonomies. Instead, additional rules such as those underlying ML2 are required to enforce adherence to a leveling mechanism, thereby revealing to users the mistakes they inadvertently commit.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

References

[1] E. Mayr, The Growth of Biological Thought: Diversity, Evolution, and Inheritance, Harvard University Press, 1982.
[2] V.A. Carvalho, J.P.A. Almeida, Toward a well-founded theory for multi-level conceptual modeling, Softw. Syst. Model. 17 (2018).
[3] F. Brasileiro, J.P.A. Almeida, V.A. Carvalho, G. Guizzardi, Expressive multi-level modeling for the semantic web, in: International Semantic Web Conference (1), in: Lecture Notes in Computer Science, vol. 9981, 2016, pp. 53–69.
[4] C. Gonzalez-Perez, B. Henderson-Sellers, A powertype-based metamodelling framework, Softw. Syst. Model. 5 (1) (2006) 72–90.
[5] B. Neumayr, K. Grün, M. Schrefl, Multi-level domain modeling with M-objects and M-relationships, in: APCCM, in: CRPIT, vol. 96, Australian Computer Society, 2009, pp. 107–116.
[6] C. Atkinson, R. Gerbig, Melanie: multi-level modeling and ontology engineering environment, in: Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards, 2012, pp. 1–2.
[7] M.A. Jeusfeld, B. Neumayr, Deeptelos: Multi-level modeling with most general instances, in: ER, in: Lecture Notes in Computer Science, vol. 9974, 2016, pp. 198–211.
[8] J. de Lara, E. Guerra, Deep meta-modelling with metadepth, in: TOOLS (48), in: Lecture Notes in Computer Science, vol. 6141, Springer, 2010, pp. 1–20.
[9] J.P.A. Almeida, C.M. Fonseca, V.A. de Carvalho, A comprehensive formal theory for multi-level conceptual modeling, in: ER, in: Lecture Notes in Computer Science, vol. 10650, Springer, 2017, pp. 280–294.
[10] C.M. Fonseca, J.P.A. Almeida, G. Guizzardi, V.A. Carvalho, Multi-level conceptual modeling: from a formal theory to a well-founded language, in: International Conference on Conceptual Modeling, Springer, 2018, pp. 409–423.
[11] F. Brasileiro, J.P.A. Almeida, V.A. Carvalho, G. Guizzardi, Applying a multi-level modeling theory to assess taxonomic hierarchietory in wikidata, in: Proc. 25th WWW, 2016.
[12] V.A. Carvalho, J.P.A. Almeida, G. Guizzardi, Using a well-founded multi-level theory to support the analysis and representation of the powertype pattern in conceptual modeling, in: International Conference on Advanced Information Systems Engineering, Springer, 2016, pp. 309–324.
[13] V.A. Carvalho, J.P.A. Almeida, C.M. Fonseca, G. Guizzardi, Multi-level ontology-based conceptual modeling, Data Knowl. Eng. 109 (2017) 3–24.
[14] G. Guizzardi, Ontological foundations for structural conceptual models (Ph.D. thesis), Telematica Instituut / CTIT, University of Twente, 2005.
[15] V.A. Carvalho, J.P.A. Almeida, A semantic foundation for organizational structures: A multi-level approach, in: 2015 IEEE 19th International Enterprise Distributed Object Computing Conference, IEEE, 2015, http://dx.doi.org/10.1109/EDOC.2015.18, 50–10.
[16] D. Nešić, M. Nyberg, Applying multi-level modeling to dataintegration in product line engineering, in: CEUR Workshop Proceedings: MODELS 2017 Satellite Events, 2017, pp. 235–242.
[17] L. Cardelli, Structural subtyping and the notion of power type, in: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1988, pp. 70–79.
[18] J. Odell, Power types, J. Object-Oriented Program. 7 (2) (1994) 8–12.
[19] D. Foxvog, Instances of instances modeled via higher-order classes, Found. Aspects Ontol. (9–2005) (2005) 46–54.
[20] OWL 2 Web Ontology Language Document Overview: W3C Recommendation 27 October 2009, OWL Working Group and others, 2009.
[21] J. Mylopoulos, Conceptual modelling and telos, in: Conceptual Modelling, Databases, and CASE: an Integrated View of Information System Development, John Wiley & Sons, 1992, pp. 49–68.
[22] C. Atkinson, T. Kühne, Model-driven development: a metamodeling foundation, IEEE Softw. 20 (5) (2003).
[23] G. Guizzardi, J.P.A. Almeida, N. Guarino, V.A. de Carvalho, Towards an ontological analysis of powertypes, in: JOWO@IJCAI, in: CEUR Workshop Proceedings, vol. 1517, CEUR-WS.org, 2015.
[24] 404: The JSON Data Interchange Format, ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, 2013.
[25] D. Jackson, Software Abstractions: Logic, Language, and Analysis, MIT press, 2012.
[26] G. Sutcliffe, The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v6.4.0, J. Automat. Reason. 59 (4) (2017) 483–502.
[27] J.P.A. Almeida, F.A. Musso, V.A. Carvalho, C.M. Fonseca, G. Guizzardi, Preserving multi-level semantics in conventional two-level modeling techniques, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 142–151.
[28] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (10) (2014) 78–85.
[29] T. Kühne, A story of levels, in: R. Hebig, T. Berger (Eds.), Proceedings of MODELS 2018 Workshops, Copenhagen, Denmark, October, 14, 2018, in: CEUR Workshop Proceedings, vol. 2245, 2018, pp. 673–682, CEUR-WS.org, http://ceur-ws.org/Vol-2245/multi_paper_5.pdf.

[30] C.M. Fonseca, ML2: an expressive multi-level conceptual modeling language (Master's thesis), Federal University of Espírito Santo, Brazil, 2017, pp. 1–93.
[31] U. Frank, Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges, Softw. Syst. Model. 13 (3) (2014) 941–962.
[32] C. Atkinson, T. Kühne, Meta-level independent modelling, in: International Workshop on Model Engineering At 14th European Conference on Object-Oriented Programming, vol. 12, 2000, p. 16.
[33] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, Ontology Library (Wonder-Web Deliverable D18), vol. 18, 2003, p. 36.
[34] OMG Unified Modeling Language (OMG UML) Version 2.5.1, Object Management Group (OMG), 2017, https://www.omg.org/spec.
[35] B. Neumayr, M.A. Jeusfeld, M. Schrefl, C.G. Schütz, Dual deep instantiation and its conceptbase implementation, in: CAiSE, in: Lecture Notes in Computer Science, vol. 8484, Springer, 2014, pp. 503–517.
[36] T. Kühne, D. Schreiber, Can programming be liberated from the two-level style: multi-level programming with DeepJava, in: Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications, 2007, pp. 229–244.
[37] T. Clark, C. Gonzalez-Perez, B. Henderson-Sellers, A Foundation for Multi-Level Modelling, CEUR, 2014.
[38] M. Igamberdiev, G. Grossmann, M. Selway, M. Stumptner, An integrated multi-level modeling approach for industrial-scale data interoperability, Softw. Syst. Model. 17 (1) (2018) 269–294.
[39] M. Selway, M. Stumptner, W. Mayer, A. Jordan, G. Grossmann, M. Schrefl, A conceptual framework for large-scale ecosystem interoperability and industrial product lifecycles, Data Knowl. Eng. 109 (2017) 85–111.
[40] S.P. Jácome-Guerrero, J. de Lara, TOTEM: Reconciling multi-level modelling with standard two-level modelling, Comput. Stand. Interfaces 69 (2020) 103390.
[41] F. Macias Gomez de Villar, A. Rutle, V. Stolz, MultEcore: Combining the best of fixed-level and multilevel metamodelling, in: CEUR Workshop Proceedings, 2016.
[42] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt, Conceptbase - a deductive object base for meta data management, J. Intell. Inf. Syst. 4 (2) (1995) 167–192.
[43] M.A. Jeusfeld, J.P.A. Almeida, V.A. Carvalho, C.M. Fonseca, B. Neumayr, Deductive reconstruction of mlt* for multi-level modeling, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, in: MODELS '20, Association for Computing Machinery, New York, NY, USA, 2020, https://doi.org/10.1145/3417990.3421410.
[44] G. Yang, M. Kifer, C. Zhao, V. Chowdhary, FLORA-2: User's manual, in: Version 0.94 (Narumigata). April, vol. 30, Citeseer, 2005.
[45] M. Igamberdiev, G. Grossmann, M. Stumptner, An implementation of multi-level modelling in F-logic, in: MULTI@ MoDELS, 2014, pp. 33–42.
[46] A. Rodríguez, F. Macías, Multilevel modelling with multecore: A contribution to the multi process challenge, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 152–163.
[47] C. Atkinson, T. Kühne, On evaluating multi-level modeling, in: MODELS (Satellite Events), 2017, pp. 274–277.
[48] C. Atkinson, R. Gerbig, T. Kühne, Comparing multi-level modeling approaches, in: MULTI@ MoDELS, 2014, pp. 53–61.
[49] M. Rosemann, P. Green, M. Indulska, A reference methodology for conducting ontological analyses, in: 23rd Int. Conf. on Conceptual Modeling, ER, Springer, 2004, pp. 110–121.
[50] J.P.A. Almeida, F.A. Musso, V.A. Carvalho, C.M. Fonseca, G. Guizzardi, Capturing multi-level models in a two-level formal modeling technique, in: International Conference on Conceptual Modeling, Springer, 2019, pp. 43–51.

Claudenir M. Fonsica is a Ph.D. student at the Free University of Bolzano and member of the Conceptual and Cognitive Modelling Research Group (CORE). He holds a MSc. Degree in Computer Sciences at the Federal University of Espírito Santo, where he developed research as member of the Ontology and Conceptual Modeling Research Group (NEMO). Claudenir has been working in the latest years on building, formalizing, and validating core ontologies, as well as developing software support for ontology-driven conceptual modeling. He is currently interested in ontology-driven conceptual modeling, multi-level modeling, and the application of domain-specific languages for ontology specification.

João Paulo A. Almeida is Associate Professor at the Federal University of Espírito Santo, Brazil, and Senior Member of the Ontology and Conceptual Modeling Research Group (NEMO). He holds a Ph.D. in Computer Science from the University of Twente, The Netherlands. For over a decade, he has been working on the application of ontologies in conceptual modeling, enterprise architecture and enterprise modeling. Particularly of interest to this project, he has a decade long experience in the development of Model-Based Engineering tools and was directly involved in the development of a number of computational tools for OntoUML. He is a senior member of the IEEE and the ACM. He serves as a member of the IEEE EDOC Conference Steering Committee and of the Advisory Board of the International Association for Ontology and its Applications (IAOA).

Giancarlo Guizzardi is a Professor of Computer Science at the Free University of Bozen-Bolzano, Italy, where he leads the Conceptual and Cognitive Modeling Research Group (CORE). He is also a Professor of Software Science and Evolution at the University of Twente, The Netherlands. He has been active for more than two decades in the areas of Ontologies, Conceptual Modeling, and Information Systems Engineering. He is currently an associate editor for the Applied Ontology journal and for Data & Knowledge Engineering and a member of a number of international journal editorial boards. He is a member of the ER Steering Committee and of the Advisory Board of the International Association for Ontology and its Applications (IAOA).

Victorio A. Carvalho is Associate Professor at the Federal Institute of Espírito Santo and member of the Ontology and Conceptual Modeling Research Group (NEMO) of the Federal University of Espírito Santo. He holds a PhD degree in Computer Science from the Federal University of Espírito Santo. His current research focuses on the combination between Ontology-Based Conceptual Modeling and Multi-level Modeling.