Multi-Level Conceptual Modeling: From a Formal Theory to a Well-Founded Language

Claudenir M. Fonseca[1], João Paulo A. Almeida[2], Giancarlo Guizzardi[1,2] and Victorio A. Carvalho[3]

[1]Free University of Bozen-Bolzano, Bolzano, Italy
[2]Federal University of Espírito Santo (UFES), Vitória, ES, Brazil
[3]Federal Institute of Espírito Santo (IFES), Colatina, ES, Brazil
cmoraisfonseca@unibz.it; jpalmeida@ieee.org;
giancarlo.guizzardi@unibz.it; victorio@ifes.edu.br;

Abstract. Subject domains are often conceptualized with entities stratified into a rigid two-level structure: a level of classes and a level of individuals which instantiate these classes. Multi-level modeling extends the conventional two-level classification scheme by admitting classes that are also instances of other classes, a feature which is key in a number of subject domains. Despite the advances in multi-level modeling 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 tackle this issue by proposing an expressive multi-level conceptual modeling language (dubbed ML2). We follow a principled approach in the design of ML2, constructing its abstract syntax as to reflect a formal 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.

Keywords: Multi-level modeling; conceptual modeling; modeling language.

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 levels 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 [5, 8, 20]. 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. A species, in its turn, is an instance of Taxonomic Rank. Other examples of multiple classification levels come from domains such as software development [13] and product types [22].

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., [2, 18, 19, 22]). 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 current modeling approaches. For example, many approaches do not support 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.

In this paper, we tackle these issues by proposing an expressive multi-level conceptual modeling language, called ML2 – Multi-Level Modeling Language. The language is aimed at multi-level (domain) conceptual modeling and is designed 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 [1]). We propose a textual syntax for ML2, which is supported by a featured Xtext-based editor in Eclipse. 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. Further, we show how ML2 incorporates rules to prevent the construction of unsound multi-level models (reflecting formal rules from MLT*).

This paper is further structured as follows: Section 2 briefly presents MLT*, which is the semantic foundation of ML2. Section 3 presents ML2's abstract and concrete syntax. Section 4 presents the related work, comparing ML2 to current multi-level techniques. Finally, Section 5 presents our final considerations.

2 MLT*: The Multi-Level Theory

Types are 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 have no possible instances, i.e., they are entities that cannot *be applied to* other entities (e.g. "John", "this apple", "my cellphone"). In the philosophical literature, types are said to be repeatable, while instance 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". Also, some of the types are both instances and classifiers of other entities, for example "Lion" classifies "Cecil" and is instance of (or is classified by) "Species". In the following examples, and before we arrive at the proposal of ML2, we use merely for illustrative purposes a notation inspired by the class and object notations of UML, using dashed arrows to represent relations that hold between the elements, with labels to denote the relation that applies (in this case *instance of*).

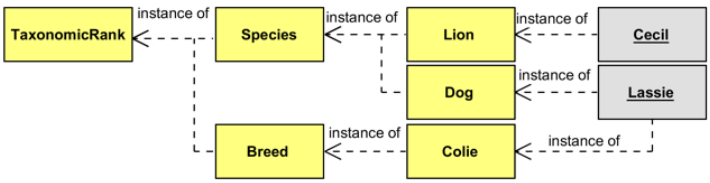

Fig. 1. A four-level instantiation chain representing a biological domain.

The theory is built up first by defining common structural relations to support conceptual modeling, starting from *specialization* between types (a transitive and reflexive relation), and *proper specialization* (which is in its turn irreflexive). 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 [7], 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").

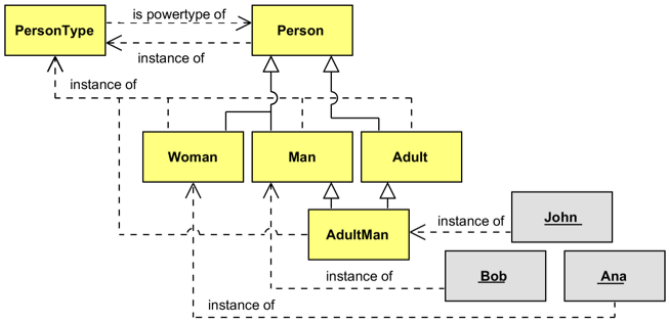

Fig. 2. PersonType and its instances.

In order to address also the notion of powertype introduced by Odell [24], 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*. Further 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.

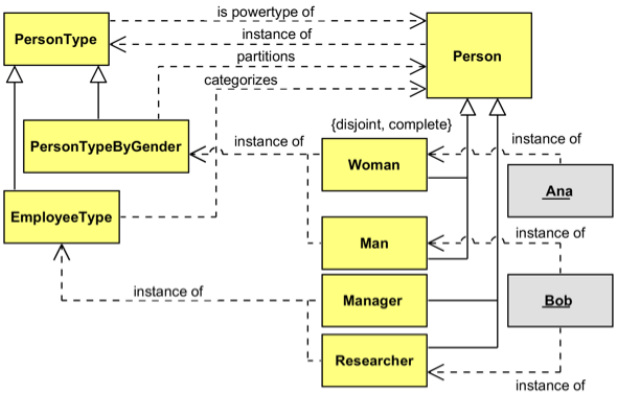

Fig. 3. Examples of the categorization and partitions relations.

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, these types are called *ordered types*. The theory explicitly accounts for orders using *basic types*. 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 the order above (e.g., "Person"). The ellipsis in that figure represents that this chain of basic types can be extended as far as demanded, given the entities involved in the captured domain. However, the formalization of MLT* does not necessitate infinite chains of basic types, allowing the description of finite models (see [1] for details).

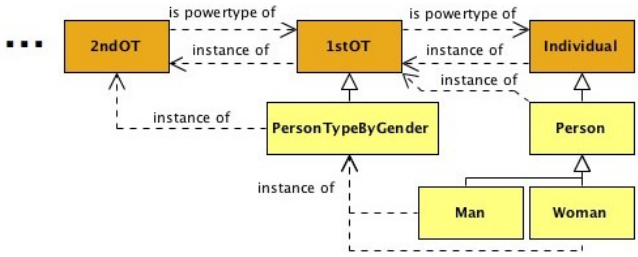

Fig. 4. Example of MLT*'s basic types.

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 include notions of "Entity", "Resource" and "Property", which are key to a number of comprehensive conceptualizations [12, 14, 21, 25]. In MLT*, types that do not conform to the stratified scheme are denominated *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 (having individual cellphones as instances). Since "BusinessAsset" has instances in different orders it is an example of a domain orderless type (Fig. 5). Finally, MLT* can also be used to describe its own categories of types resulting in the model shown in Fig. 5.

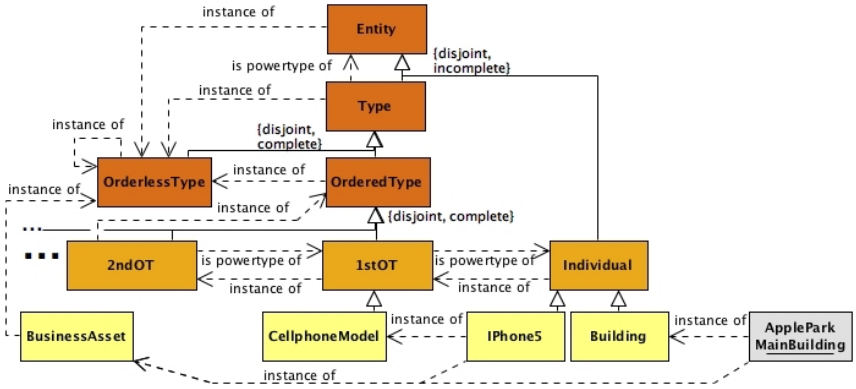

Fig. 5. MLT* basic scheme extended by a domain example.

MLT* formally defines a number of structural relations and rules to govern such relations. Some of these rules concern the nature of the instantiation relation: whenever instantiation involves solely ordered types, it is irreflexive, anti-symmetric and anti-transitive, leading to a strict stratification of types. However, when involving orderless types, there are situations in which instantiations can be reflexive, symmetric or transitive. Table 1 summarizes the logical properties and rules of types involved in MLT* structural relations, which arise out of the axioms and theorems of the theory [1].

Table 1. Summary of constraints on MLT* relations.

Relation (t → t')	Domain	Range	Constraint	Properties
Specialization	Orderless	Orderless	if *t* and *t*' are ordered types, they must be at the same type order	Reflexive, anti-symmetric, transitive
	Ordered	Orderless		
	Ordered	Ordered		
Proper Specialization	Orderless	Orderless		Irreflexive, anti-symmetric, transitive
	Ordered	Orderless		
	Ordered	Ordered		
Powertype	Orderless	Orderless	*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, anti-symmetric, anti-transitive
	Ordered	Ordered		
Categorization	Orderless	Orderless	*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, anti-symmetric, non-transitive
Disjoint Categorization	Ordered	Orderless		
	Ordered	Ordered		
Complete Categorization	Orderless	Orderless		Irreflexive, anti-symmetric, anti-transitive
Partitioning	Ordered	Ordered		

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 in any degree of generality. The development of ML2 has been based on the Xtext framework and provides a featured Eclipse editor[1], including model validation capabilities and compatibility with EMF-based technologies.

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: *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. Both classes and individuals may declare the instantiation of multiple classes. Specialization (proper) and the other structural relations of the theory are considered for classes. For a class that categorizes another

[1] The ML2 editor can be found at https://github.com/nemo-ufes/ML2-Editor

class, a categorization type should be defined to reflect which variant of the categorization relation applies.

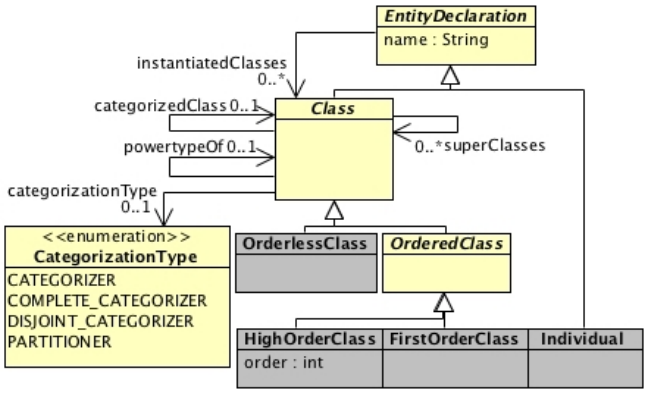

Fig. 6. Entities and classes in ML2.

Some of the rules of the theory are directly reflected in the representation of cardinality constraints. For example, given the formal definition of powertype in MLT*, a class can only be the powertype of at most one other class. All the other MLT* rules (including those that apply to the instantiation relation and those present in Table 1) are reflected in the validation functionality of the editor coded in Xtend (a high-level programming language for the Java Virtual Machine) and presented lively. The syntax of ML2[2] is inspired by traditional OO languages and applies a collection of keywords aiming at enhancing the readability of its models. The statements for entity declaration follow a common pattern, varying the available structural relations for each type of entity. Fig. 7 revisits the examples presented so far using ML2. Note that a namespace mechanism is supported with modules. Unlike other multi-level languages, ML2 modules are not order bound (e.g., like MetaDepth's notion of "model" [19] that restricts the order of contained entities).

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. 8 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 by JSON's specification [10].

[2] The language's grammar is available at https://github.com/nemo-ufes/ml2-grammar.

```
module example.model {
  orderless class BusinessAsset;
  order 2 class CellphoneModel;
  class IPhone5 : CellphoneModel, BusinessAsset;
  order 2 class PersonType isPowertypeOf Person;
  order 2 class EmployeeType specializes PersonType categorizes Person;
  class Person : PersonType;
  class Manager : EmployeeType specializes Person;
  class Researcher : EmployeeType specializes Person;
  individual Bob : Person, Researcher;
}
```

Fig. 7. Examples of entity declarations in ML2.

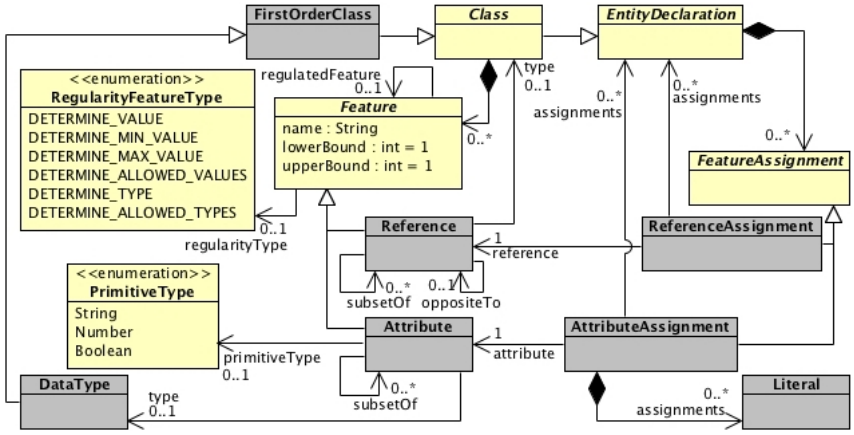

Fig. 8. Features and assignments in ML2.

Fig. 9 presents an example of usage of features in an ML2 model. This model expands the one in Fig. 7 by explicitly capturing the cross-level reference *owns* between "Enterprise" and "BusinessAsset" and adding some other entities. Note that ML2 does not require exhaustive feature assignment (see "IPhone5" without an assignment for *belongsTo*). However, cardinality constraints are checked for every present assignment.

ML2 also accounts for a special kind of feature called *regularity feature* (see [8,15]). 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. Fig. 10 presents a modification of Fig. 9 illustrating the usage of regularity features.

```
orderless class BusinessAsset {
     ref belongsTo : Enterprise isOppositeTo owns
};
class Enterprise {
     ref owns : [0..*] BusinessAsset isOppositeTo belongsTo
};
class Building;
order 2 class CellphoneModel categorizes Cellphone;
class Cellphone {
     screenSize : Number
};
class IPhone5 : CellphoneModel, BusinessAsset specializes Cellphone{
     ref belongsTo = Apple
};
individual AppleParkMainBuilding : Building, BusinessAsset;
individual Apple : Enterprise {
     ref owns = { AppleParkMainBuilding , IPhone5 }
};
individual MyIPhone : IPhone5{
     screenSize = 4
};
```

Fig. 9. Examples of features in ML2.

```
order 2 class CellphoneModel categorizes Cellphone {
     reglarity instancesScreenSize : Number determinesValue screenSize
     regularity ref compatibleProcessorModel : ProcessorModel
          determinesType installedProcessor
};
class Cellphone {
     screenSize : Number
     ref installedProcessor : Processor
};
class IPhone5 : CellphoneModel specializes Cellphone {
     instancesScreenSize = 4
     ref compatibleProcessorModel = A6
};
order 2 class ProcessorModel categorizes Processor;
class Processor;
class A6 : ProcessorModel specializes Processor;
individual Processor01 : A6;
individual MyIPhone : IPhone5 {
     screenSize = 4
     ref installedProcessor = Processor01
};
```

Fig. 10. Examples of regularity features in ML2.

ML2 foresees six types of regularity features. In the case above, values of *instancesScreenSize* determines 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) and to determine the set of allowed values for a feature (e.g., to model that a phone model has either 16 or 32GB of internal storage capacity). Additionally, a regularity feature can 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.

Fig. 10 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, if present, are checked for conformance.

4 Related Work

In this section, we position ML2 with respect to the existing work in multi-level representation approaches regarding a number of features, *which are all supported by ML2*. We consider the following multi-level modeling approaches: Melanee, M-Objects, MetaDepth and DeepTelos. Table 2 summarizes our evaluation of the various modeling approaches: a plus sign ('+') indicates support for the feature, a minus sign ('-') indicates no support, and plus/minus ('+/-') indicates partial support.

Table 2. Multi-level modeling techniques comparison.

Representation Features	Melanee	M-Objects	MetaDepth	DeepTelos
F1: represents entities of multiple classification levels	+	+	+	+
F2: allows an arbitrary number of classification levels	+	+	+	+
F3: defines guiding principles for organization of models	+	+	+	+/-
F4: represents types that defy a stratified classification scheme	+/-	–	+/-	+
F5: represents rules to govern instantiation of related types	–	–	–	+/-
F6: allows domain relations between entities in various levels	–	+	+	+
F7: represents domain features and feature assignments	+	+	+	+
F8: relates features of entities in different levels	+	–	+	–

Melanee [2] is a tool that supports multi-level modeling founded on the notions of *strict metamodeling*, *clabject* and *potency*. It is based on the idea of defining *clabjects* and *fields* (attributes and slots) within the levels of a strict stratified scheme (i.e., *strict metamodeling* [4]) 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 [22], 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 [22]. 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 scheme 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 [23], 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 [19] 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 close to the one adopted in Dual-Deep Instantiation [23]), allowing the representation of domain relations between clabjects at different instantiation levels. Therefore, MetaDepth supports all the features Melanee supports, and further supports F6.

Finally, DeepTelos is a knowledge representation language that approaches multi-level modeling with the application of the notion of "most general instance (MGI)" [18]. In [17], 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 admits relations between types in different levels, thus, supporting F6. DeepTelos supports the attribution of values to features of types (F7). However, its account for attributes does not include any support to explain the relationship between attributes of entities in different classification levels, thus, not supporting F8.

5 Final Considerations

In this paper, we have presented the ML2 multi-level conceptual modeling language. We have approached the design of ML2 with a careful consideration of the conceptualization of types in classification schemes that transcend a rigid two-level structure. The language harnesses the conceptualization formalized in MLT*, reflecting the theory's definitions in its constructs and syntactical constraints. 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. Further, 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 is one of the distinctive features of ML2 (see [1] for axiomatization and reference to a specification in the Alloy language [16]).

During this research, we investigated a number of multi-level representation techniques reported in the literature, focusing on their capability of capturing different intended multi-level scenarios. We have observed that multi-level approaches often opt for one of two extremes: (i) to consider all classes to be orderless (or similarly to ignore the organization of elements into stratified orders, the case of DeepTelos, what is referred to as a level-blind approach in [3]), or (ii) to consider all classes to be strictly stratified (e.g., in the case of Melanee and MetaDepth). Approaches that opt for (i) are able to represent all types ML2 can capture, however, fail to provide rules to guide the use of the various structural relations (e.g., instantiation). As shown in [6], this lack of guidance has serious consequences on resulting models quality. Approaches that opt for (ii) do not support the representation of a number of important abstract notions, including those very general notions that we use to articulate multi-level domains (such as "types", "clabjects", "entities") (these abstract notions are key to ML2 being able to model the upper portion of the UFO foundational ontology [14], see ML2 models produced for it in [11] including a response to the so-called "Bicycle Challenge").

The combination of both approaches in the design of ML2 places it in a unique position in multi-level modeling approaches. On top of that, the present approaches for multi-level modeling also strive in order to accommodate the implications of the dual nature of "clabjects" regarding the representation of features and feature assignments. Besides ML2, only MetaDepth and Melanee were able to express related features in different levels and cross-level references. Although, both of them support the representation of only one of the various types of regularity attributes supported in ML2.

A few knowledge representation approaches (e.g., DeepTelos [18] and Cyc [12]) have also drawn distinctions between orderless and ordered types. However, Telos does not provide rules for the various structural relations, including instantiation and specialization. In its turn, Cyc, arguably the world's largest and most mature knowledge base nowadays, employs a conceptual architecture for types similar to MLT*'s distinctions of sorts of entities (see Fig. 5). Additionally, this architecture also includes instantiation and specialization rules [12]. However, these rules are not incorporated in some representation language and no deep characterization mechanism is provided in Cyc.

Future work concerning ML2 includes the development of transformations from ML2 into the Semantic Web approach discussed in [5], the development of an integrated constraint language to further increase the expressiveness of the language, and the investigation of a suitable visual syntax to accompany the textual syntax.

Acknowledgements. This work is partially supported by CNPq (grants number 407235/2017-5, 312123/2017-5 and 312158/2015-7), CAPES (23038.028816/2016-41), FAPES (69382549) and FUB (OCEAN Project).

References

1. Almeida, J.P.A., Fonseca, C.M., Carvalho, V.A.: A Comprehensive Formal Theory for Multi-Level Conceptual Modeling. ER Conferece 2017. Valencia, Spain (2017).
2. Atkinson, C., Gerbig, R.: Melanie: multi-level modeling and ontology engineering environment. Proceedings of the 2nd International Master Class on Model-Driven Engineering Modeling Wizards - MW '12, ACM Press, New York, USA (2012).

3. Atkinson, C., Gerbig, R., Kühne, T.: Comparing Multi-Level Modeling Approaches. Proceedings of the 1st International Workshop on Multi-Level Modelling (2014).
4. Atkinson, C., Kühne, T.: Meta-level Independent Modeling. In International Workshop "Model Engineering" (in conjunction with ECOOP'2000). Cannes, France, p. 16 (2000).
5. Brasileiro, F., Almeida, J.P.A., Carvalho, V.A., Guizzardi, G.: Expressive Multi-level Modeling for the Semantic Web. In: Groth P. et al. (eds) The Semantic Web – ISWC 2016. ISWC 2016. Lecture Notes in Computer Science, vol 9981. Springer, pp. 53-69 (2016).
6. Brasileiro, F. et al.: Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata. In Proceedings of the 25th International Conference Companion on World Wide Web. pp. 975–980. Geneva, Switzerland (2016).
7. Cardelli, L.: Structural subtyping and the notion of powertype. Proceedings of the 15th ACM Symposium of Principles of Programming Languages, pp. 70–79 (1988).
8. Carvalho, V.A., Almeida, J.P.A.: Toward a well-founded theory for multi-level conceptual modeling. Software & Systems Modeling, Springer Berlin Heidelberg, pp. 1-27 (2016).
9. Clark, T., Gonzalez-Perez, C., Henderson-Sellers, B.: Foundation for multi-level modelling. CEUR Workshop Proceedings, 1286, pp. 43–52 (2014).
10. ECMA: The JSON Data Interchange Format. 1st Edition. Available at: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (2013).
11. Fonseca, C. M.: ML2: an expressive multi-level conceptual modeling language. Dissertation (master's in informatics) - Federal University of Espírito Santo, Brazil (2017).
12. Foxvog, D.: Instances of instances modeled via higher-order classes, Foundational Aspects of Ontologies, (9–2005), pp. 46–54. Available at: http://www.uni-koblenz.de/fb4/publikationen/gelbereihe/RR-9-2005.pdf (2005).
13. Gonzalez-Perez, C., Henderson-Sellers, B.: A powertype-based metamodelling framework. Software & Systems Modeling, vol. 5, pp. 72–90 (2006).
14. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. 1st ed. The Netherlands (2005).
15. Guizzardi, G. et al.: Towards an Ontological Analysis of Powertypes. Proceedings of the Joint Ontology Workshops 2015, 1517 (2015).
16. Jackson, D.: Software Abstractions: Logic, Language and Analysis. MIT Press (2006).
17. Jarke, M. et al.: ConceptBase - A deductive object base for meta data management, Journal of Intelligent Information Systems, 4(2), pp. 167–192 (1995).
18. Jeusfeld, M. A., Neumayr, B.: DeepTelos: Multi-level Modeling with Most General Instances. 35th Int'l Conf., ER 2016. Springer International Publishing, pp. 198–211 (2016).
19. de Lara, J., Guerra, E.: Deep Meta-modelling with MetaDepth, Proceedings of the 48th International Conference, TOOLS 2010, Málaga, Spain (2010).
20. Mayr, E.: The Growth of Biological Thought: Diversity, Evolution, and Inheritance. The Belknap Press (1982).
21. Mylopoulos, J.: Conceptual modeling and Telos. Conceptual Modelling, Databases, and CASE: an Integrated View of Information System Development, John Wiley & Sons, New York, New York. Edited by P. Loucopoulos and R. Zicari. Wiley, pp. 49–68 (1992).
22. Neumayr, B., Grun, K., Schrefl, M.: Multi-level domain modeling with m-objects and m-relationships. 6th Asia-Pacific Conf. on Conceptual Modelling, vol. 96, pp. 107-116 (2009).
23. Neumayr, B. et al.: Dual Deep Instantiation and Its ConceptBase Implementation. Int'l Conf. on Advanced Information Systems Engineering (CAiSE), pp. 503–517 (2014).
24. Odell, J.: Power types. Journal of Object-Oriented Programming, 7(2), pp. 8-12 (1994).
25. W3C: OWL 2 Web Ontology Language Document Overview. Available at: http://www.w3.org/TR/2009/REC-owl2-overview-20091027/ (2009).