Capturing Multi-Level Models in a Two-Level Formal Modeling Technique

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

[1] Ontology & Conceptual Modeling Research Group (NEMO), Federal University of Espírito Santo (UFES), Brazil
jpalmeida@ieee.org, fernandomusso14@gmail.com
[2] Federal Institute of Espírito Santo (IFES), Colatina, Brazil
victorio@ifes.edu.br
[3] Conceptual and Cognitive Modeling Research Group (CORE), Free University of Bozen-Bolzano, Italy
cmoraisfonseca@unibz.it, giancarlo.guizzardi@unibz.it

Abstract. Conceptual models are often built with techniques that propose a strict stratification of entities into two classification levels: a level of types (or classes) and a level of instances. Multi-level conceptual modeling extends the conventional two-level scheme by admitting that types can be instances of other types, giving rise to multiple levels of classification. Nevertheless, the vast majority of tools and techniques are still confined to the two-level scheme, and hence cannot be used for multi-level models directly. We show here how a multi-level model in ML2 can be transformed into a two-level specification in the formal modeling technique Alloy, thereby leveraging the Alloy analyzer to multi-level models.

Keywords: Multi-Level Modeling, model transformation

1 Introduction

Conceptual modeling is usually undertaken by capturing invariant aspects of the entities in a subject domain, which is supported in most conceptual modeling approaches through constructs such as "classes" and "types", reflecting the use of "kinds", "categories" and "sorts" in accounts of a subject domain by subject matter experts. In the conventional two-level representation scheme, a conceptual model is stratified into two levels of entities: a level of types (or classes) and a level of instances (or individuals). The level of types captures invariants that apply exclusively to the level of instances. In this scheme, the subject matter can be understood as consisting of individuals, and the purpose of the conceptual model is to establish which structures of individuals are admissible according to some (shared) conceptualization of the world [16].

The two-level scheme, however, reveals its limitations whenever we are interested in invariants about categories themselves [8], i.e., whenever categories of categories are part of the domain of inquiry. For example, in the biological taxonomy domain [2, 6], living beings are classified according to biological 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) [22]. 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. Thus, to describe the conceptualization in this domain, one needs to represent entities of different (yet related) classification levels, such as specific living beings (Cecil), types of living beings (Lion), types of types of living beings (Species, Animal Species). In fact, classification levels can be added as required, e.g., Taxonomic Rank classifies the types of types of living beings. Other examples of multiple classification levels can be found in organizational roles, software engineering [15] and product types [23].

The need to represent entities in such domains led to what is currently termed "multi-level modeling" [4, 23]. Techniques for multi-level modeling must provide modeling concepts to deal with types in various classification levels and address the relations that may occur between those types. Moreover, they must account for types behaving as instances and, as such, respecting invariants and holding values for properties they exemplify (in other words, types in a multi-level model have two facets: a "class" or type facet, and an "object" or instance facet [4]).

Despite the benefits of multi-level modeling, multi-level mechanisms pose a challenge to the reuse of existing two-level techniques and tools. In this paper, we demonstrate how multi-level models can be accommodated in a conventional two-level language. We propose a systematic transformation of multi-level models represented in the ML2 Multi-Level Modeling Language [13] into conventional Alloy [17] specifications, following a transformation pattern that is based on the reification of the instance facet of a type and its systematic linking with the type facet. This allows us to leverage model simulation and verification support originally designed with a conventional two-level scheme to multi-level conceptual models.

The remainder of this paper is organized as follows: Section 2 discusses a classical workaround employed when we are confined to two levels of classification, namely, the powertype pattern. Section 3 briefly presents the multi-level modeling language we adopt here (ML2). Section 4 discusses how multi-level models are represented in a corresponding two-level specification in Alloy. Section 5 presents some conclusions.

2 The Classical Two-Level Workaround – The Powertype Pattern

Let us consider the biological taxonomy domain as a paradigmatic example of a multi-level domain [2, 6]. We are interested in this domain not only in capturing features of certain organisms (e.g., its weight), but also features of types of organisms and their properties. For example, a Species (like other taxa) is named by a Person (the Lion species was named by Carl Linnaeus) and can be attributed a conservation status. Further, being a member of a certain species, an organism has certain features in virtue of being a member of the species. For example, all lions are warm blooded, while all frogs are cold blooded.

In the conventional two-level approach, entities in the domain have to be classified either as classes (or types) or as instances. Strictly speaking, there is no room for meta-types such as Species or meta-meta-types such as Taxonomic Rank. Workarounds are available as discussed in [19, 20], but these often introduce accidental complex-

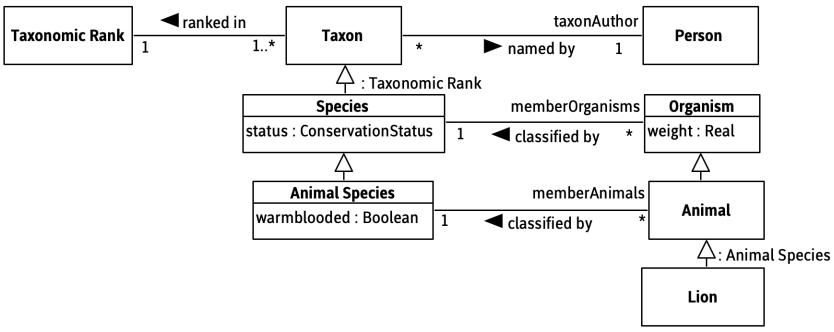

Fig. 1. The powertype pattern with a regular user-defined association

ity. For example, an early approach that has aimed to accommodate multiple domain levels within two modeling levels is the powertype pattern proposed by [24]. In this pattern, all types are treated as regular classes, and a "base type" (such as `Organism`, `Taxon`) is related to a "powertype" (such as `Species`, `Taxonomic Rank`), through a user-defined (and regular) association (such as `classified by`, `ranked in`). See Figure 1 for a model capturing this scenario using UML's support for powertypes.

This workaround creates a number of difficulties, some of which are discussed in [19,20]. First of all, a modeler needs to handle explicitly two notions of instantiation, a native one provided by the modeling technique (and thus between classes and instances) and another that corresponds to the user-defined association (`classified by`, `ranked in`). In the case of the latter, since it is a regular user-defined association, no support for its instantiation semantics is provided by the modeling technique, and hence, instantiation semantics needs to be *emulated* manually by the modeler. The pattern is based on the duplication of the instances of the powertype: this is because they must be admitted both at the instance level (e.g., `Lion` as an instance of `Species`, carrying values for `taxonAuthor`, `conservation status`, `warmblooded`) and at the same time at the class level (e.g., `Lion` as a specialization of `Organism`). The management of the duplicated entities—although key to the pattern—is left to the model user.

3 The ML2 Multi-Level Modeling Language

The root of the problem discussed in the previous section is that two-level languages fail to recognize classes as instances of other (meta)classes [3,14]. This has motivated some of us in the past to propose a Multi-Level Modeling Language (ML2) [13], following work on theoretical foundations for Multi-Level Modeling (MLT [8,10] and MLT* [1]).

The language provides support to the specification of properties of individuals, their types (the so-called first-order types), second-order types (whose types are first-order types) and so on. Further, the language incorporates notions of powertype in the literature (including Odell's [24] and Cardelli's powertypes [7]). The language supports a number of features typical of multi-level modeling techniques, some of which are exemplified here; we refer the reader to [12,13] for further details.

In Listing 1.1, we employ ML2 to revisit the example from Figure 1 presenting how a language that is not limited to the two-level scheme captures the multi-level notions.

Listing 1.1. Fragment of ML2 model involving `Lion` and `AnimalSpecies`

```
1  order 2 class AnimalSpecies categorizes Animal {
2   regularity instancesAreWarmBlooded: Boolean
3    determinesValue isWarmBlooded
4   ref taxonAuthor : Person };
5  class Organism { weight : Number};
6  class Animal specializes Organism { isWarmBlooded :Boolean };
7  class Lion : AnimalSpecies specializes Animal {
8   ref taxonAuthor = CLinnaeus
9   instancesAreWarmBlooded = true };
10 class Person { name : String };
11 individual CLinnaeus : Person { name = 'Carl Linnaeus' };
12 individual Cecil : Lion { isWarmBlooded = true };
```

Individuals (entities that are not classes) are marked `individual`. Simple `class` declarations capture first-order classes (e.g., `Animal`, `Lion` and `Person`), whose instances are individuals (e.g., `Cecil` and `CLinnaeus`). Higher-order classes are declared by using a `order` modifier (e.g., `AnimalSpecies` in Line 1 is a second-order class).

Differently from the conventional powertype approach, rather than relying on domain relations with no specialized semantics (such as `ranked in` or `classified by` in Fig. 1), ML2 enables the expression of instantiation between classes, which is represented by a colon. Given the specialized semantics, constraints enforce that high-order classes can only have as instances classes at the order immediately below. Standard modeling features of specialization, attributes and references are also present in the language, and both attributes and references may have values assigned for their instances. For example, `'Carl Linnaeus'` is the `name` of `CLinnaeus` (an individual `Person`) and `CLinnaeus` is the `taxonAuthor` of `Lion`. In addition, `regularity` features are used to represent *deep instantiation* [2], when the attributes of a higher-order type affect entities at lower levels. By assigning `instancesAreWarmBlooded=true` in the declaration of `Lion` (Line 7), the value of `isWarmBlooded` is regulated for all its instances, including `Cecil`. The `determinesValue` keyword (Line 3) specifies the sort of regulation and the regulated feature. (See [12, 13] for other kinds of regulation supported in ML2.)

The powertype pattern semantics is supported with the so-called categorization relations between classes. All instances of a class that `categorizes` another (in an adjacent lower order) are (direct or indirect) specializations of the categorized class. Thus, by declaring that `AnimalSpecies categorizes Animal` (Line 1), all instances of `AnimalSpecies` (such as `Lion`) specialize `Animal` (a constraint enforced by ML2).

4 A Systematic Two-Level Solution and its Alloy Implementation

Similarly to the powertype pattern discussed in Section 2, we reify the instance facet of a type in our two-level representation scheme. However, differently from the powertype pattern, the instance and type facets are systematically linked to each other. The result is that the expression of multi-level constraints becomes possible, and, at the same time,

the technique-native support for instantiation, attribute assignment and specialization is preserved. We establish the following *representation rules* for a two-level scheme: (i) each ML2 class is represented as a regular class (capturing its type facet); (ii) each ML2 class (at any order) is reified at the instance level (capturing its instance facet); (iii) native instantiation of a class is reflected in explicit instantiation links between a class instance and the reified class being instantiated; (iv) in addition to instantiation, specialization is reified as links between the reified classes. In order to ensure that the multi-level semantics is preserved, (v) all classes specialize classes in a top-level library corresponding to ML2 notions (`Type`, its ordered specializations, `Individual`).

The established correspondence results in the representation schema shown in Figure 2. The topmost layer that corresponds to ML2 notions is represented in white. It is extended by introducing `Taxonomic Rank`, `Animal Species`, `Animal` and `Lion` corresponding to the classes in Listing 1.1 (`Person` was omitted due to space constraints). The figure also shows a possible instance level. As required, it includes reified instances of `Type` (e.g., `lionReified`) that correspond to the various classes on the left-hand side of the figure (e.g., `lionReified` corresponds to the `Lion` class). It also shows a possible instance of `Lion`, called `cecil`. The object level shows `cecil` linked to `lionReified` through the `instance of` association. As discussed in Section 2, there is a purposeful duplication of the `Lion` entity in order to reveal its instance facet at the object level through `lionReified`.

We employ this principled approach to implement a transformation from ML2 into a conventional two-level language dubbed Alloy [17]. Alloy is a formal two-level language designed to support lightweight formal techniques for simulation and verification of model specifications. For the implemented transformation, an Alloy specification of the top-level ML2 layer (in fact, MLT* [9]) is imported in all of the generated Alloy specifications, serving as the topmost layer for our transformation. We shall explore how each representation rule discussed above is applied in the ML2 to Alloy transformation. In order to illustrate the application of the transformation, we use the example discussed in Listing 1.1 and the corresponding Alloy fragment in Listing 1.2.

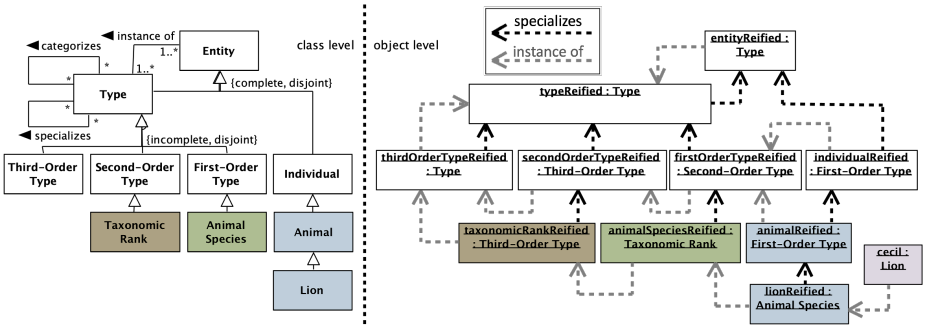

Fig. 2. Reification example.

Listing 1.2. Fragment of Alloy specification involving Lion

```
1  sig Animal in Organism { isWarmblooded: Boolean }
2  one sig AnimalReified in Species {}
3  fact AnimalReifiedDefinition {
4    all e: Entity | e in AnimalReified iff
5      (all e': Entity | iof[e',e] iff e' in Animal) }
6  sig Lion in Animal {}
7  one sig LionReified in AnimalSpecies {} {
8    taxonAuthor = CarlLinnaeus
9    instancesAreWarmBlooded = true }
10 fact LionReifiedDefinition {
11   all e: Entity | e in LionReified iff
12     (all e': Entity | iof[e',e] iff e' in Lion) }
13 one sig Cecil in Lion {}
14 one sig CarlLinnaeus in Person {} { name = "Carl␣Linnaeus" }
15 sig AnimalSpecies in Species {
16   instancesAreWarmBlooded: Boolean }
17 one sig AnimalSpeciesReified in Order2Type {}
18 fact AnimalSpeciesReifiedDefinition {
19   all e: Entity | e in AnimalSpeciesReified iff
20     (all e': Entity | iof[e',e] iff e' in AnimalSpecies) }
21 fact AnimalSpeciesCategorizesAnimal {
22   categorizes[AnimalSpeciesReified,AnimalReified] }
23 fact instancesAreWarmbloodedRegulatesisWarmblooded {
24   all x:Animal |
25     x.isWarmblooded = (x.iof).instancesAreWarmblooded }
```

Concerning rule (i), class declarations, native specialization, typing of relations and attributes, are all supported by Alloy directly. This allows for a direct representation of classes, e.g., in lines 1, 6 and 15 (`Animal`, `Lion` and `AnimalSpecies`) and respective specializations (`Species`, `Organism` and `Person` were omitted due to space constraints). Attribute and reference declarations are supported directly (see `isWarmBlooded` in line 1 and `instancesAreWarmBlooded` in line 16).

Concerning rule (ii), we include the instance facet of classes into the specification. Due to the absence of native support for instance declaration in Alloy, we make use of singletons, see lines 2, 7 and 17 (`AnimalReified`, `LionReified`, `AnimalSpeciesReified`), a solution we also employ for the representation of individuals in lines 13 and 14 (`Cecil` and `CarlLinnaeus`). These instance facets (for classes and individuals alike) are the holders of any assignments and are used to declare categorizations (see lines 21 and 22 for `AnimalSpecies` categorizing `Animal`).

Concerning rule (iii), instantiation is reified through the `iof` predicate that is defined in the top-level library for MLT* imported in all specifications. This rule further binds the different facets of classes by enforcing that, whenever an entity natively instantiates a class (as denoted with the keyword `in`) it also has an explicit instantiation link with the reified class (and vice-versa). See lines 3–5, 10–12 and 18–20 which respectively bind `Animal` and `AnimalReified`, `Lion` and `LionReified`, and `AnimalSpecies` and `AnimalSpeciesReified`, following the same transformation pattern.

Specialization links between reified classes (iv) do not have to be declared explicitly, as they are a logical consequence of rule (iii) and the definition of specialization in the top-level library. Rule (v) results in declaration of specializations towards the library classes (`Order1Type`, `Order2Type`). Finally, regularity attributes result in the declaration of a regulation fact (lines 23–25). In this case, all animals of a certain species are warmblooded (or not) as defined by the species `instancesAreWarmBlooded` attribute.

The transformation was implemented on top of the Eclipse-based ML2 editor `https://github.com/nemo-ufes/ML2-Editor`. The full implementation of the transformation and the listing of the original ML2 model and corresponding Alloy specification can be found in `https://github.com/nemo-ufes/ml2-to-alloy`.

5 Final Considerations

In this paper, we have shown how to capture ML2 multi-level models in a two-level representation scheme in Alloy. The approach builds up on the powertype pattern, by reifying the instance facet of types. Differently from the powertype pattern, we systematically link type and instance facets. As a result, a two-level technique such as Alloy can be directly used in the simulation and verification of multi-level models. By incorporating a formally defined (MLT*-based) top-level library, we can enforce consistent usage of ML2 notions, from instantiation and order stratification to regularity features.

The literature presents a number of approaches for dealing with multi-level domains within the limitation of two-level schemes. We observe among these three typical cases: (i) the application of the powertype pattern that captures relations between classes of different orders [15]; (ii) the reification of classes as entities that capture their instance facets [18]; (iii) and the application of metamodeling strategies consisting on stacking two-level models to represent higher-order classes as metaclasses [21]. Each of these approaches exhibit some limitations in the representation of multi-level domains: (i) powertype-based approaches fail to link instance and type facets and do not capture the specialized instantiation semantics underlying categorization relations, (ii) simple reification approaches fail to provide level structuring mechanisms which are require to rule out unsound models [5]; and (iii) the stacking (or cascading) approach places corresponding entities in distinct two-level models preventing the expression of cross-level relations (such as `named by` between `Species` and `Person`).

We expect the transformation approach we discuss here to be applicable to other two-level representation languages with minor effort. We are working on a first-order logic formalization of the approach, in order to generalize it to other techniques beyond Alloy. It is part of our present research agenda to explore the implications of our approach to the representation of multi-level models in OWL-DL and UML, following the work reported in [6, 11].

Acknowledgments

This work has been partially supported by CNPq (407235/2017-5, 312123/2017-5), 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. In: Proc.36th ER (2017)
2. Atkinson, C., Kühne, T.: Model-Driven Development: a Metamodeling Foundation. IEEE Software **20**(5) (2003)
3. Atkinson, C., Kühne, T.: Meta-Level Independent Modelling. In: Proc.14th ECOOP (2000)
4. Atkinson, C., Kühne, T.: The Essence of Multilevel Metamodeling. In: Proc.4th UML (2001)
5. Brasileiro, F., Almeida, J.P.A., Carvalho, V.A., Guizzardi, G.: Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata. In: Proc.25th WWW (2016)
6. Brasileiro, F., Almeida, J.P.A., Carvalho, V.A., Guizzardi, G.: Expressive Multi-level Modeling for the Semantic Web. In: Proc.15th ISWC (2016)
7. Cardelli, L.: Structural Subtyping and the Notion of Power Type. In: Proc.15th POPL (1988)
8. Carvalho, V.A., Almeida, J.P.A.: Toward a Well-Founded Theory for Multi-Level Conceptual Modeling. Software & Systems Modeling **17** (2018)
9. Carvalho, V.A., Almeida, J.P.A., Fonseca, C.M., Guizzardi, G.: Extending the foundations of ontology-based conceptual modeling with a multi-level theory. In: Conceptual Modeling. pp. 119–133. Springer (2015)
10. Carvalho, V.A., Almeida, J.P.A., Fonseca, C.M., Guizzardi, G.: Multi-level ontology-based conceptual modeling. Data Knowl. Eng. **109**(C), 3–24 (May 2017)
11. Carvalho, V.A., Almeida, J.P.A., Guizzardi, G.: Using a Well-Founded Multi-Level Theory to Support the Analysis and Representation of the Powertype Pattern in Conceptual Modeling. In: Proc.28th CAiSE (2016)
12. Fonseca, C.M.: ML2: An Expressive Multi-Level Conceptual Modeling Language. Master's thesis, Federal University of Espírito Santo (2017)
13. Fonseca, C.M., Almeida, J.P.A., Guizzardi, G., Carvalho, V.A.: Multi-level Conceptual Modeling: From a Formal Theory to a Well-Founded Language. In: Proc.37th ER (2018)
14. Foxvog, D.: Instances of Instances Modeled via Higher-Order Classes. In: FOnt 2005 Workshop, Proc. 28th KI (2005)
15. Gonzalez-Perez, C., Henderson-Sellers, B.: A Powertype-Based Metamodelling Framework. Software & Systems Modeling **5** (2006)
16. Guizzardi, G.: On Ontology, Ontologies, Conceptualizations, Modeling Languages, and (Meta)Models. Frontiers in artificial intelligence and applications **155** (2007)
17. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2012)
18. Kimura, K., et al.: Practical Multi-level Modeling on MOF-compliant Modeling Frameworks. In: Proc.2nd MULTI Workshop (2015)
19. Kühne, T., Schreiber, D.: Can Programming Be Liberated from the Two-level Style: Multi-level Programming with Deepjava. In: Proc.22nd OOPSLA (2007)
20. Lara, J.D., Guerra, E., Cuadrado, J.S.: When and How to Use Multilevel Modelling. ACM Transactions on Software Engineering and Methodology (TOSEM) **24** (2014)
21. Macías, F., Rutle, A., Stolz, V.: MultEcore: Combining the Best of Fixed-Level and Multi-level Metamodelling. In: Proc. 3rd MULTI Workshop (2016)
22. Mayr, E.: The Growth of Biological Thought: Diversity, Evolution, and Inheritance. Harvard University Press (1982)
23. Neumayr, B., Grün, K., Schrefl, M.: Multi-Level Domain Modeling with M-Objects and M-Relationships. In: Proc.6th APCCM (2009)
24. Odell, J.: Power Types. Journal of OO Programming **7** (1994)