Preserving Multi-Level Semantics in Conventional Two-Level Modeling Techniques

João Paulo A. Almeida[1], Fernando A. Musso[1], Victorio A. Carvalho[2],
Claudenir M. Fonseca[3], Giancarlo Guizzardi[3]
[1]*Ontology & Conceptual Modeling Research Group (NEMO), Federal University of Espírito Santo (UFES), Vitória, Brazil*
[2]*Federal Institute of Espírito Santo (IFES), Colatina, Brazil*
[3]*Conceptual and Cognitive Modeling Research Group (CORE), Free University of Bozen-Bolzano, Bolzano, Italy*
jpalmeida@ieee.org, fernandomusso14@gmail.com, victorio@ifes.edu.br,
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 (individuals, classes, metaclasses, metametaclasses, and so on). As a result, multi-level models capture not only invariants about individuals, but also invariants about types themselves, which become regular elements of the domain of inquiry (first-class citizens). Despite the benefits of the multi-level approach, the vast majority of tools for conceptual modeling are still confined to the two-level scheme, and hence cannot accommodate multi-level entities. This paper proposes a transformation of multi-level to two-level models that preserves the semantics of the original multi-level model. We employ the systematic reification of the instance facet of a class and its linking to the type facet. The feasibility of the approach is demonstrated by a transformation of ML2 (multi-level) models to Alloy (two-level) specifications.**

***Index Terms*—multi-level modeling, model transformation, multi-level theory, multi-level modeling language**

1. Introduction

Conceptual modeling is usually undertaken by capturing invariant aspects of 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, where the level of types captures invariants that apply exclusively to the level of individuals. 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 [1].

The two-level scheme, however, reveals its limitations whenever categories of categories are part of the domain of inquiry. In this setting, we are interested not only in invariants about individuals but also about categories themselves [2]. For example, in the military domain, experts often refer to "vessel types" or "ship classes" in their accounts. Instances of `Vessel Type`, such as `Cargo Ship`, `Submarine`, `Supercarrier` or `Nimitz-class Aircraft Carrier` are themselves types, instantiated by individual vessels. For instance, the `USS Abraham Lincoln` (an aircraft carrier of the US Navy) is an instance of both the `Supercarrier` and `Nimitz-class Aircraft Carrier` types. In their turn, `Supercarrier` and `Nimitz-class Aircraft Carrier` are instances of `Vessel Type`. Thus, to describe the conceptualization in this domain, one needs to represent entities of different (but nonetheless related) classification levels, such as specific ships out there in a mission, types of ships, and (even) types of types of ships (`Vessel Type`, `Submarine Type`, `Cargo Ship Type`). Other examples of multiple classification levels come from domains such as that of organizational roles, software engineering [3], biological taxonomy [4] and product types [5].

The need to support the representation of subject domains dealing with multiple classification levels has given rise to what has been referred to as "multi-level modeling" [5], [6]. Techniques for multi-level conceptual modeling must provide modeling concepts to deal with types in various classification levels and 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. Despite the modeling expressiveness gained from the support provided by multi-level techniques, they pose a significant challenge to frequently used representation languages that adopt a strict two-level divide (such as, e.g., Alloy, OWL-DL) or that have limited multi-level modeling capabilities (as is the case of UML as shown in [7]).

This work has been partially supported by CNPq (407235/2017-5, 312123/2017-5), CAPES Finance Code 001 (23038.028816/2016-41), FAPES (69382549) and FUB (OCEAN Project).

Over the last years, some of us worked out a foundational theory for multi-level modeling (dubbed MLT*) [8]. Our investigation was motivated by the lack of (language-independent) foundations for multi-level modeling. MLT* embodies those conceptual notions that are key to the representation of multi-level models, such as the existence of entities that are simultaneously types and instances, the iterated application of instantiation across an arbitrary number of levels, the possibility of defining attributes and values at the various type levels, etc. In this paper, we employ MLT* to devise a well-founded solution to systematically transform multi-level models into strict two-level languages while fully preserving its original features. This paper advances on a recent short paper [9], on which we explore the transformation of models represented in ML2 [10] (a language that incorporates the rules of MLT* into its syntax) into Alloy (a two-level formal language [11]), by generalizing and formally describing the systematic emulation of the semantics of multi-level mechanisms in the two-level scheme. Additionally, we exemplify the power of emulating original model semantics in two-level language by exploring the simulation of ML2 models in Alloy.

The remainder of this paper is organized as follows: Section 2 discusses workarounds which are required when we are confined to two levels of classification, focusing on the powertype pattern. Section 3 presents the fragment of MLT* which is employed in our solution in Section 4. Section 5 discusses the ML2 to Alloy transformation formally defining and exemplifying the required steps. Section 6 discusses related approaches, and finally, Section 7 presents some concluding remarks.

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 [4], [12]. The biological taxonomy for living beings classifies them 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`) [13]. `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`. In this domain, we are interested not only in capturing features of certain organisms (e.g., Cecil's cause of death), but also of types of organisms (species), 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 [14], [15], but these often introduce accidental complexity that is not related to the complexity of dealing with higher-order types, but complexity that emerges from the specific workaround. For example, an early approach that has aimed to accommodate multiple domain levels within two modeling levels is the powertype pattern proposed by [16]. 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.

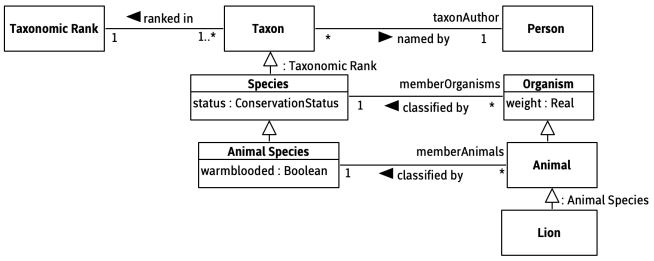

Figure 1. The powertype pattern with a regular user-defined association

This workaround creates a number of difficulties, some of which are discussed in [14], [15]. 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` or `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`, `population`, `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.

Some other problems stem from the reliance on "generalization sets" to indicate that a subclass is an instance of a powertype. For a classifier to be considered a "powertype" in UML, it must be related to a "generalization set". Thus, in UML, the powertype pattern can only be applied when specializations of the base type are explicitly modeled (otherwise there would be no generalization set). We consider this undesirable as it rules out simple models such as one defining `Species` as a "powertype" of `Organism`, without forcing the modeler to "hardcode" specific instances for `Species`.

3. Background: The MLT* multi-level theory

Before we provide a systematic two-level solution, we present here a formal multi-level modeling theory some of us proposed in [8]. Here, we discuss the theory's topmost fragment, which is adopted in Section 4 as part of the two-level solution.

The theory ranges over all possible types and individuals, which together constitute the entities we are interested in (a1). Every entity (type or individual alike) is related to one or more types through a primitive instance of relation (or for short iof). Since types can themselves be related to other types through iof, this enables chains of instantiation of arbitrary lengths. Those entities that never play the role of type in instantiation are the individuals (a2). We assume that the theory is only concerned with types with non-trivially false *intensions*, i.e., with types that have possible instances in the conceptualization. Because of this, for every type there is a possible entity that instantiates it (a3).

a1 $\forall x(\mathsf{Entity}(x))$

a2 $\forall x(\mathsf{Individual}(x) \leftrightarrow \neg\exists y(\mathsf{iof}(y,x)))$

a3 $\forall x(\mathsf{Type}(x) \leftrightarrow \exists y(\mathsf{iof}(y,x)))$

We establish that types are ultimately grounded on individuals. Thus a super-relation (iof') is defined including all pairs such that $\mathsf{iof}(x,y)$, but 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 $\forall t(\mathsf{Type}(t) \rightarrow \exists x(\mathsf{Individual}(x) \wedge \mathsf{iof'}(x,t)))$

Specialization between types is defined as usual, i.e., a type specializes a supertype whenever all its instances are also instances of the supertype (a5). *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 (a6).

a5 $\forall t_1,t_2(\mathsf{specializes}(t_1,t_2) \leftrightarrow$
$\mathsf{Type}(t_1) \wedge \forall x(\mathsf{iof}(x,t_1) \rightarrow \mathsf{iof}(x,t_2)))$

a6 $\forall t_1,t_2(\mathsf{properSpecializes}(t_1,t_2) \leftrightarrow$
$\mathsf{specializes}(t_1,t_2) \wedge \neg\mathsf{specializes}(t_2,t_1)))$

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

a7 $\forall t_1,t_2(\mathsf{Type}(t_1) \wedge \mathsf{Type}(t_2) \rightarrow (t_1 = t_2 \leftrightarrow \forall x(\mathsf{iof}(x,t_1) \leftrightarrow$
$\mathsf{iof}(x,t_2))))$

Using the basic framework outlined above, 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 [16]. 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. 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). A categorizes relation between types was defined to reflect Odell's notion of powertype [16]. 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. 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`) [12].

a8 $\forall t_1,t_2(\mathsf{isPowertypeOf}(t_1,t_2) \leftrightarrow$
$\mathsf{Type}(t_1) \wedge \forall t_3(\mathsf{iof}(t_3,t_1) \leftrightarrow \mathsf{specializes}(t_3,t_2)))$

a9 $\forall t_1,t_2(\mathsf{categorizes}(t_1,t_2) \leftrightarrow$
$\mathsf{Type}(t_1) \wedge \forall t_3(\mathsf{iof}(t_3,t_1) \rightarrow \mathsf{properSpecializes}(t_3,t_2)))$

4. A Systematic Two-Level Solution

Our solution involves incorporating the fragment of MLT* discussed in Section 3 in a top-level library of the two-level model. Since it reflects MLT*, explicit support for key notions such as *instantiation* and *specialization* is provided with no two-level constraint. Further, relations between types (*is powertype of*, *categorization*) are supported in order to express key multi-level constraints.

Similarly to the powertype pattern discussed in section 2, the instance facet of a type is reified in this approach. 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.

4.1. The supporting representation scheme

For the purposes of creating the top-level library and establishing the correspondence between the formalization and the conceptual model, some representation rules are established, as follows. First, we assume that each class defined with a conceptual modeling technique corresponds to a one-place predicate, and each binary association corresponds to a two-place predicate. Using this correspondence, we can establish the top layer of our conceptual model (Figure 2), introducing the classes `Entity`, `Type` and `Individual` to reflect the Entity, Type and Individual predicates defined in Section 3, along with associations for instantiation, proper specialization, powertype and categorization also corresponding to predicates defined in Section 3, namely, iof, properSpecialization, isPowertypeOf and categorizes (improper specialization was omitted).

Second, for every native specialization declaration[1] we introduce a constraint in accordance with the following *formula schema*, in which the proxy predicates *subclass* and *superclass* are replaced by the suitable domain predicates:

a10 $\forall x(subclass(x) \rightarrow superclass(x)) \wedge \exists y(superclass(y) \wedge \neg subclass(y))$

For example, in the case of the classes `Lion` and `Animal`, and `Animal` and `Individual`, the following constraints are implied:

a11 $\forall x(\mathsf{Lion}(x) \rightarrow \mathsf{Animal}(x)) \wedge \exists y(\mathsf{Animal}(y) \wedge \neg \mathsf{Lion}(y))$

a12 $\forall x(\mathsf{Animal}(x) \rightarrow \mathsf{Individual}(x)) \wedge \exists y(\mathsf{Individual}(y) \wedge \neg \mathsf{Animal}(y))$

In the case of the classes `Animal Species` and `Type`, the following constraint is implied (a13). `Animal Species` specializes `Type` since its instances are types themselves according to the theory.

a13 $\forall x(\mathsf{AnimalSpecies}(x) \rightarrow \mathsf{Type}(x)) \wedge \exists y(\mathsf{Type}(y) \wedge \neg \mathsf{AnimalSpecies}(y))$

The established correspondence results in the representation schema shown in Figure 3, which incorporates the model in Figure 2 as topmost layer, and extends it introducing `Animal`, `Animal Species` and `Lion`. The figure also shows a possible object level, in this particular case with an instance of `Animal Species`, which was called `lionReified` to avoid a name clash with the `Lion` class, and an 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, at the class level and at the object level (`lionReified`), revealing its class and its instance facets.

Native instantiation[2]—such as that between `cecil` and the class `Lion`, and between `lionReified` and `Animal Species`—correspond to predication using the corresponding classes. Constants represent the entities being predicated (e.g., $\mathsf{Lion}(\mathsf{cecil})$ and $\mathsf{AnimalSpecies}(\mathsf{lionReified})$).

Note that first-order logics is employed, and there is no quantification over predicates or predication applied to predicates themselves (instead, predication is over individuals and reified types). This means that our formalization strategy reflects the content of a conceptual model in a stratified two-level technique, which a clear segregation of predicates on the one hand (classes in the two-level scheme), and non-predicative elements of the domain of quantification (objects in the two-level scheme).

1. We use this term to distinguish between the two sorts of specialization, with *native* referring to the specialization as supported by the two-level conceptual modeling technique, as opposed to the `properSpecializes` relation in the theory.

2. Again, we distinguish between instantiation as supported by the two-level conceptual modeling technique and the `iof` relation in the theory.

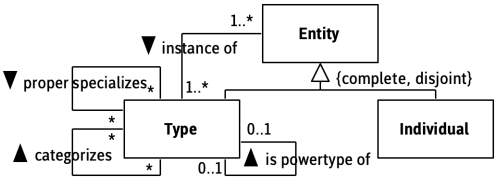

Figure 2. The topmost layer.

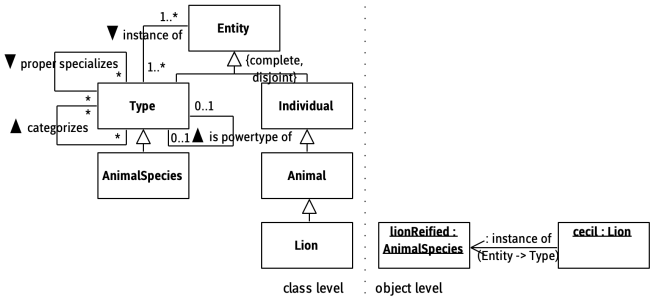

Figure 3. Reification example.

4.2. The missing link between instance and type facets

Thusfar, we are able to recreate the approach underlying the powertype pattern. However, similarly to the powertype pattern, we have not yet provided mechanisms to manage those reified instances of types at the object level. As discussed in Section 2, the management of the reified entities and their correspondence to subclasses of the base type — although key to the pattern — is left to the model user. Because of this, there is no guarantee that only one reification of `Lion` exists, and further that every instance of `Lion` (like `cecil`) is related to that reified type through iof.

Because of this, for every type whose instance facets are reified, we introduce a constraint effectively linking instance and type facets with the following *formula schema*:

a14 $\forall x(class(x) \leftrightarrow \mathsf{iof}(x, reifiedclass))$

Reading (14) from left to right, we have that every (native) instance of a class is required to be related to the specific reified class through iof. Reading (14) from right to left, we have that every entity related to the reified class through iof instantiates (natively) the corresponding class. Applying the schema to `Lion` and `lionReified`, we have that:

a15 $\forall x(\mathsf{Lion}(x) \leftrightarrow \mathsf{iof}(x, \mathsf{lionReified}))$

The schema can be applied to any type to be reified, including the types in the topmost layer. If we apply this schema to all other types in Figure 3, we have that:

a16 $\forall x(\mathsf{Entity}(x) \leftrightarrow \mathsf{iof}(x, \mathsf{entityReified}))$

a17 $\forall x(\mathsf{Type}(x) \leftrightarrow \mathsf{iof}(x, \mathsf{typeReified}))$

a18 $\forall x(\mathsf{Individual}(x) \leftrightarrow \mathsf{iof}(x, \mathsf{individualReified}))$

a19 $\forall x(\mathsf{Animal}(x) \leftrightarrow \mathsf{iof}(x, \mathsf{animalReified}))$

a20 $\forall x(\mathsf{AnimalSpecies}(x) \leftrightarrow \mathsf{iof}(x, \mathsf{animalSpeciesReified}))$

4.3. Consequences of the representation scheme

The proposed strategy results in the reflection of the class level at the object level. This mechanism is similar to introspection in programming languages with an explicit meta-object protocol. An important requirement for this reflection at the object level is that the reified entities are related according to their native counterparts. This applies to native instantiation, but also applies to the specialization relation.

It follows directly from the linking schema (a14) that the reified class is the only entity that can be linked in that particular way to the corresponding class predicate. Formally, theorem schema (t1) holds. For instance, (t2) follows from (a15).

t1 $\forall t(\forall x(class(x) \leftrightarrow \mathsf{iof}(x,t)) \rightarrow (t = reifiedclass))$

t2 $\forall t(\forall x(\mathsf{Lion}(x) \leftrightarrow \mathsf{iof}(x,t)) \rightarrow (t = \mathsf{lionReified}))$

Further, it follows from the linking schema (a14) and the definition of proper specialization (a6) that the reified classes corresponding to sub- and superclasses are adequately linked by properSpecializes. That is, whenever the reified class proper specializes another reified class, their counterparts follow the native specialization scheme (a10) (and vice-versa). Formally we have the theorem schema (t3). For instance, (t4) follows from (a15), (a19) and (a6).

t3 $\mathsf{properSpecializes}(reifiedsubclass, reifiedsuperclass) \leftrightarrow (\forall x(subclass(x) \rightarrow superclass(x)) \wedge \exists y(superclass(y) \wedge \neg subclass(y)))$

t4 $\mathsf{properSpecializes}(\mathsf{refiedLion}, \mathsf{reifiedAnimal}) \leftrightarrow (\forall x(\mathsf{Lion}(x) \rightarrow \mathsf{Animal}(x)) \wedge \exists y(\mathsf{Animal}(y) \wedge \neg \mathsf{Lion}(y)))$

Figure 4, which expands on Figure 3 incorporating more concepts inspired by Figure 1, shows the result of this strategy, with the mirroring of the class level at the object level (for simplicity, only the most specific `instance of` links are shown; further, transitive `proper specializes` links are omitted). We employ here colors to highlight the presence of multiple classification levels for the domain classes, leaving the white color for library entities, including `First-Order Type`, `Second-Order Type` and `Third-Order Type`, which account for a dynamic part of the generated top-most library that we shall further explore in Section 4.5. Note that native instantiation in the object diagram matches with corresponding `instance of` links. The same can be said of native specialization and `proper specializes` links. The reified instantiation relation is multi-level, chaining `cecil` to `Animal`, `AnimalSpecies` and beyond, what cannot be said of native instantiation.

4.4. Representing multi-level modeling mechanisms

Given the correspondence between the two-level and the reified multi-level model, we are now in a position to enforce the semantics of the powertype pattern simply by asserting the required relation between refied powertype and basetype. In the example depicted in Figure 4, we can assert further that $\mathsf{categorizes}(\mathsf{AnimalSpecies}, \mathsf{Animal})$. The semantics of categorizes along with the reflection scheme enforces that all instances of `Animal Species` (such as `lionReified`) are linked to the reified basetype (`animalReified`) through proper specialization, and, given the other constraints, that the class corresponding to `lionReified` (`Lion`) also natively specializes `Animal`. Importantly, different from the UML support for powertypes discussed in Section 2, this approach does not force the modeler to enumerate subclasses of `Animal` in order to establish that all instances of `Animal Species` specialize the base type `Animal`. This is a consequence of the `categorizes` link between `Animal Species` and `Animal`.

Concerning attributes of types, native language support can be used to attribute values and links to the refied type (in other words, the type's instance facet). For example, one could say that `lionReified` was named by `Carl Linnaeus` (instance of `Person`), and that its conservation status is `VULNERABLE` (using a `ConservationStatus` enumeration following the categories defined by the International Union for Conservation of Nature)[3]. Note that relations crossing levels are naturally supported in the approach, since even the types present in the model are reified as instances.

This support can also be used to represent a multi-level phenomenon called *deep instantiation* [4], 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], [19]. 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 `instancesAreWarmblooded` of `AnimalSpecies` regulates the attribute `isWarmblooded` of `Animal`. In the case of `lionReified`, `instancesAreWarmblooded=true` and hence all its instances (such as `cecil`) must have `isWarmblooded=true`. In order to ensure that, whenever this sort of regulation is called for, the following formula schema is used:

a21 $\forall x,y(regulatedclass(x) \wedge regulatingclass(y) \wedge \mathsf{iof}(x,y) \rightarrow (regulatedattr(x) = regulatingattr(y)))$

a22 $\forall x,y(\mathsf{Animal}(x) \wedge \mathsf{AnimalSpecies}(y) \wedge \mathsf{iof}(x,y) \rightarrow (\mathsf{isWarmblooded}(x) = \mathsf{instancesAreWarmblooded}(y)))$

3. As shown in [18], one can use the different values of an enumeration to derive subtypes of a type t associated to that enumeration. In particular, a mutable attribute such as `status` can be used to derive *dynamic* higher-order types (such as `VulnerableSpecies`) in the spirit of [19].

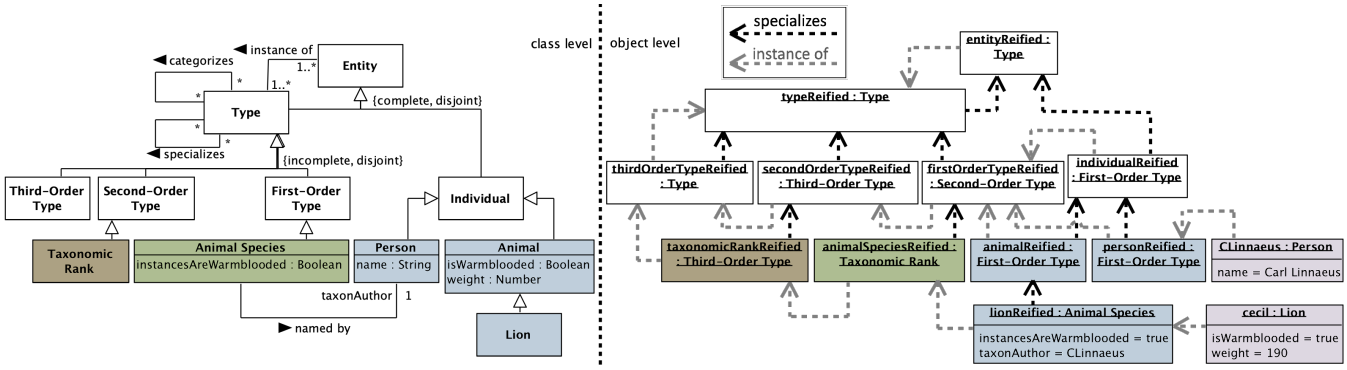

Figure 4. Reflecting the class level at the object level.

4.5. Stratification – addressing level blindness

A key requirement to a multi-level approach is to define principles (rules) for the organization of entities into "levels". Such principles are not provided by the so-called "level-blind" approaches [20] (such as UML), which results in problematic taxonomic structures as shown with empirical evidence from Wikidata in [21].

Leveling is supported in MLT* by identifying some types as "ordered": (i) *First-order types* are those types whose instances are individuals. Examples include `Person`, `Lion`, `Animal`, `Organism`; (ii) *Second-order types* are those types whose instances are first-order types. Examples include `Taxon`, `Species`, `Animal Species`; (iii) *Third-order types* are those types whose instances are second-order types (`Taxonomic Rank`), and so on, to cope with an arbitrary number of levels [2].

Figure 4, when revisiting the top-level library of Figure 2, incorporates these ordered types. Assigning domain types an order makes it possible to identify unsound multi-level structures, with stratification rules that reflect theorems of MLT* (see [8]). For example, *specialization* (proper or not) cannot cross level boundaries, *instance of* only relates entities of adjacent orders, in a strictly stratified scheme, and *is powertype of* and *categorizes* can only be applied between types in adjacent levels.

5. Transforming ML2 Multi-Level Models to Alloy Specifications

A text-based multi-level modeling language called ML2 was designed using MLT* [10]. ML2's metamodel reflects MLT*'s top-layers, and the language incorporates the rules of MLT* into its syntax. Given the strong relation between ML2 and MLT*, a transformation from ML2 models to a conventional two-level method such as Alloy [11] follows quite closely the representation strategy discussed in Section 4. In this section, we revisit the ML2-Alloy transformation worked out in [9], highlighting how it exemplifies the generalized transformation approach discussed in Section 4.

First of all, an Alloy specification of the top-level MLT* layer is imported in all of the generated Alloy specifications. Class declarations, native specialization, typing of relations and attributes, are all supported by Alloy directly. The formula schema for linking reified types and their class facets (a14) is used for each type in the ML2 model. Regularity attributes are supported following the regulation schema (a21). Finally, ML2 declarations concerning *categorization* and *powertype* relations become declarations that reuse the predicates defined in the MLT* Alloy specification.

Listing 1 shows an ML2 model of biological taxonomy reflecting the example presented in Figure 3. It includes the definition of third, second and first-order classes — `TaxonomicRank` (Line 1); `AnimalSpecies` (Line 2); `Person`, `Animal` and `Lion` (Lines 5–10) — and uses the categorization and regularity mechanisms discussed in Section 4.4 (Lines 2–3). It also declares `Lion` as instance of `Animal Species` (along with its properties), `AnimalSpecies` as instance of `TaxonomicRank` and two individuals, `Cecil` and `CLinnaeus`.

Listing 1. Example ML2 model about biological taoxnomies.

```
1  order 3 class TaxonomicRank;
2  order 2 class AnimalSpecies : TaxonomicRank
   completeCategorizes Animal {
3   regularity instancesAreWarmblooded :
    Boolean determinesValue isWarmblooded
4   ref namedBy : Person };
5  class Person { name : String };
6  class Animal {
7   weight : Number
8   isWarmblooded : Boolean };
9  class Lion : AnimalSpecies specializes
   Animal
10  { instancesAreWarmblooded = true
11  ref namedBy = CLinnaeus };
12 individual CLinnaeus : Person { name = 'Carl
    Linnaeus' };
13 individual Cecil : Lion;
```

Listing 2 shows the generated Alloy specification involving `Lion` (but not including the top-level library). Line 23 shows a signature `Lion` corresponding to the homonymous class which is declared *natively* to be a specialization of `Animal`. Line 24 corresponds to the reification

of Lion; since Alloy does not support instance specification directly, a singleton signature is employed. Values for atttributes of LionReified (taxonAuthor and instancesAreWarmblooded) are settled in lines 25–26. Lines 27–28 correspond to the linking axiom (a15). Lines 29–30 implement the regulation of isWarmBlooded corresponding to a22. Individuals are represented using singletons (Lines 31–33). Listing 2 also contains the specification for AnimalSpecies. Lines 8 and 11–12 follow the pattern of reification discussed for Lion above. Lines 9–10 declare AnimalSpecies to categorize Animal (relying on the reified counterparts).

Listing 2. Fragment of Alloy specification for the ML2 biological taxonomy example.

```
1  sig TaxonomicRank in Order2Type {}
2  one sig TaxonomicRankReified in Order3Type
   {}
3  fact TaxonomicRankReifiedDefinition {
4   all e: Entity | e in TaxonomicRankReified
    iff (all e': Entity | iof[e',e] iff e' in
    TaxonomicRank) }
5  sig AnimalSpecies in Order1Type {
6   instancesAreWarmblooded: Boolean,
7   namedBy: Person }
8  one sig AnimalSpeciesReified in
   TaxonomicRank {}
9  fact AnimalSpeciesCompleteCategorizesAnimal
   {
10  compCategorizes[AnimalSpeciesReified,
    AnimalReified] }
11 fact AnimalSpeciesReifiedDefinition {
12  all e: Entity | e in AnimalSpeciesReified
    iff (all e': Entity | iof[e',e] iff e' in
    AnimalSpecies) }
13 sig Person in Individual { name: String }
14 one sig PersonReified in Order1Type {}
15 fact PersonReifiedDefinition {
16  all e: Entity | e in PersonReified iff (all
     e': Entity | iof[e',e] iff e' in Person) }
17 sig Animal in Individual {
18  weight: Int,
19  isWarmblooded: Boolean }
20 one sig AnimalReified in Order1Type {}
21 fact AnimalReifiedDefinition {
22  all e: Entity | e in AnimalReified iff (all
     e': Entity | iof[e',e] iff e' in Animal) }
23 sig Lion in Animal {}
24 one sig LionReified in AnimalSpecies {}{
25  instancesAreWarmblooded = true
26  namedBy = CLinnaeus }
27 fact LionReifiedDefinition {
28  all e: Entity | e in LionReified iff (all e
    ': Entity | iof[e',e] iff e' in Lion) }
29 fact
   instancesAreWarmbloodedRegulatesisWarmblooded

30  { all x: Animal | x.isWarmblooded = (x.iof)
    .instancesAreWarmblooded }
31 one sig CLinnaeus in Person {}{ name = "Carl
   ␣Linnaeus" }
32 one sig Cecil in Lion {}
33 fact disjointIndividuals { disjoint[
   CLinnaeus,Cecil] }
```

Having the two-level Alloy specification enables the modeler to further investigate the original multi-level conceptual model by means of simulations and formal verification. For example, by adding the fragment shown in Listing 3 are able to generate instances of the ML2 model (line 1), such as the one presented in Figure 5, and check the theorem (t4) (lines 2–4), presented in Section 5, within a certain numbers of entities (called scope in Alloy). When executing the check declaration for a scope of 20, the Alloy tool informs the modeler that no counterexamples are found, supporting the validity of (t4).

Listing 3. Simulating and verifying Alloy models

```
1 run {} for 14
2 check {
3   all t : Entity | ( all x : Entity | x in
    Lion iff iof[x,t] ) implies t in
    LionReified
4 } for 20
```

Figure 5 shows an example of model simulation. The generated model can be examined to validate the model. The presentation configurations of this simulation mask some entities (some reified classes of the top-level library) and some MLT* relations (non-proper specialization and powertyping) to facilitate the readability by focusing on instantiation, specialization and feature assignment. Here, native instantiation is represented by the classifiers present within each entity between parenthesis, while the reified instantiation and specialization appear as arcs connecting entities. Note here the color scheme with colors identifying different classification levels, while library entities are identified in white boxes. Most entities reflect reifications of classes and individuals we have declared in ML2 holding values for the features each instantiate, with exception of Entity 6 and Entity 10, which were automatically generated by Alloy and point to the possibility of other instances of AnimalSpecies and Animal, respectively, in that configuration. Observe that these generated entities follow the constraints of the ML2 model, with value of isWarmblooded in Entity 10 being properly regulated by instancesAreWarmblooded in Entity 6, for example. This kind of model simulation serves to show the modeler the consequences of the choices in the ML2 model.

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.

6. Related Work

Gonzalez-Perez and Henderson-Sellers discuss in [3] the nature of “powertypes”. Similar to us, the authors recognize that instances of a powertype are instances and classes at the same time and propose the explicit representation of both facets, using a notational convention. They point out that such approach advances the traditional two-level approach by allowing the dynamic creation of subtypes of the partitioned type, which is also achieved in our approach. Differently from us, the authors do not explore the link between the two facets, and do not discuss other multi-level mechanisms.

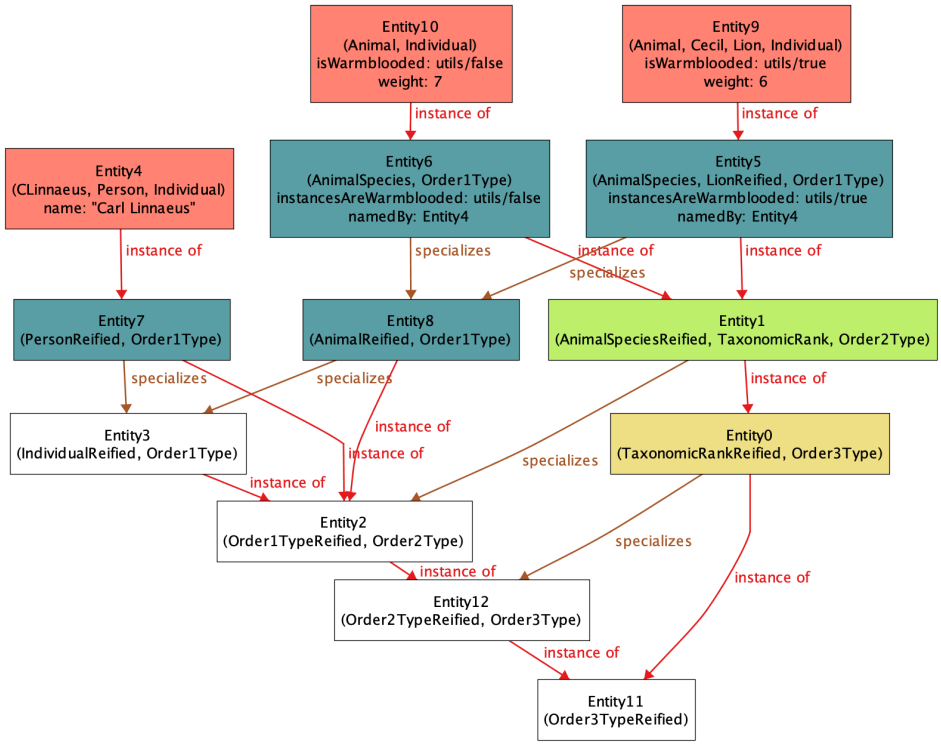

Figure 5. Alloy simulation example.

In [18], Halpin discusses the problem of addressing higher-order modeling aspects in traditional modeling frameworks that are restricted by a first-order semantics. As put by the author: *"As the move to higher-order logic may add considerable complexity to the task of formalizing and implementing a modeling approach, it is worth investigating whether the same practical modeling objectives can be met while staying within a first-order framework"*. He discusses some arguments for considering higher-order types in conceptual modeling semantics including: (i) to allow one to think of instances of certain categorization types (e.g. `AccountType`, `CarModel`) as being types themselves; and (ii) to directly represent relations and attributes that cross-levels (e.g., a `CarModel` is associated to a number of instances of `ColorChoice`. When choosing a particular `Car` of a given `CarModel`, a customer chosen feature can be a `ColorChoice`). He proposes the representation of categorization types as regular instances (of first-order types). The subtypes of a categorized type are then related to these instances by a *is of* relation that is: asymmetric, intransitive, obey mandatory and uniqueness constraints (e.g., every instance `Account` is of a unique instance of `AccountType`), as well as what he terms *local homogeneity* (e.g., only instances of Account can be related by *is of* to instance of AccountType). Moreover, regarding (ii), Halpin proposes replacing the otherwise level-crossing relation by a pair of first-order relations with constraints between them. One should notice that the semantics of *is of* relation for Halpin correspond to the asymmetric and intransitive aspects of *iof* in MLT* for "ordered" types [8] plus the definition established in (a14). However, our approach goes beyond these minimal constraints of correspondence between reified types and instances and defines the semantics of several relations to be established between reified types [2], [8]. Further, our approach combines stratification rules for "ordered" types with the possibility of representing "orderless" types, which are required in a number of modeling scenarios (see [8]). Regarding (ii), although discussing that particular case, Halpin does not generalize a solution to constraints of that sort. In our framework, his solution can be seen by as a special case of the regulated attributes scheme as discussed in Section 4.4[4].

OWL [22] enables multi-level modeling by applying the notion of contextual semantics [23], often referred to as *punning*. It allows the declaration of features of both instance and type facets within the same model. However, as discussed in [12] these two facets are treated independently in reasoning to avoid higher-order semantics. Differently from our approach, punning does not explore the link between instance and type facets. The "independence" between the two facets leads to non-intuitive interpretations. For instance, consider the following statements: (i) `Harry` is an instance of `Golden Eagle`, and; (ii) `Golden Eagle` is the same as `Aquila chrysaetos`. Statement (i) treats `Golden Eagle` as a class, while statement (ii) treats `Golden Eagle` as an

4. Although the regulation scheme discussed in Section 4.4 was focused on unary predicates, in [10], there are a number of alternative regulation schemes to generalize the formula.

instance. In OWL, it is impossible to infer that `Harry` is an instance of `Aquila chrysaetos`, which violates our intuition with respect to the multi-level model.

Kimura et al. [24] explore how to represent multi-level models on MOF-compliant (two-level) modeling frameworks. They reify the "definition" and the "instance" facets of elements, and establish OCL constraints to emulate some aspects of the semantics of the multi-level models (e.g., to emulate instantiation and specialization). They do not discuss variants of the powertype pattern, deep characterization or level stratification.

Similarly, Macías et al. [25] cope with the limitation of a two-level modeling framework (in their case EMF) to represent multi-level models. Their approach is based on the representation of multi-level models by cascading the two-level scheme in a stack of models. The only relation between entities at different levels is instantiation. As discussed at length in [2] this precludes the representation of important multi-level phenomena (such as domain relations that cross levels, variants of the powertype pattern), a criticism which is applicable to other multi-level approaches (see [2]).

7. Conclusions

In this paper, we have shown a generalized approach to transform multi-level models in a two-level representation scheme preserving the original semantics. The approach builds up on the powertype pattern, by systematically reifying a type's instance facet *and* linking type and instance facets systematically. The approach is able to leverage mechanisms such as deep characterization and level stratification to the two-level technique. To demonstrate the approach's feasibility, a transformation of ML2 models to Alloy specifications was designed. This transformation enables simulation and verification of (ML2) multi-level models through two-level (Alloy) specifications.

The use of a formally-verified semantic foundation is one of the distinctive features of ML2 when compared to other multi-level modeling approaches in the literature [10]. Here, we put this semantic foundation to direct use, by importing the MLT* formalization as a top-layer of our two-level models.

We expect the representation approach we discuss here to be applicable to other two-level representation languages with minor effort. The use of first-order logics to capture the general scheme for a corresponding two-level representation can support us in this task. Nevertheless, specificities of particular two-level techniques should still be addressed. 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, Ecore and UML, following our work in [7], [12].

Finally, ML2 is similar to other multi-level languages that rely on notions such as "clabjects" [26], [27] and m-objects [5] and recognize the duality of type and instance facets without the need for explicit reification of these facets. We believe that our solution could be adapted to transform multi-level models in some of these other multi-level techniques while preserving their semantics.

References

[1] G. Guizzardi, "On Ontology, Ontologies, Conceptualizations, Modeling Languages, and (Meta)Models," *Frontiers in artificial intelligence and applications*, vol. 155, 2007.

[2] V. A. Carvalho and J. P. A. Almeida, "Toward a Well-Founded Theory for Multi-Level Conceptual Modeling," *Software & Systems Modeling*, vol. 17, 2018.

[3] C. Gonzalez-Perez and B. Henderson-Sellers, "A Powertype-Based Metamodelling Framework," *Software & Systems Modeling*, vol. 5, 2006.

[4] C. Atkinson and T. Kühne, "Model-Driven Development: a Metamodeling Foundation," *IEEE Software*, vol. 20, no. 5, 2003.

[5] B. Neumayr, K. Grün, and M. Schrefl, "Multi-Level Domain Modeling with M-Objects and M-Relationships," in *Proc.6th APCCM*, 2009.

[6] C. Atkinson and T. Kühne, "The Essence of Multilevel Metamodeling," in *Proc.4th UML*, 2001.

[7] V. A. Carvalho, J. P. A. Almeida, and G. Guizzardi, "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.

[8] J. P. A. Almeida, C. M. Fonseca, and V. A. Carvalho, "A Comprehensive Formal Theory for Multi-level Conceptual Modeling," in *Proc.36th ER*, 2017.

[9] J. P. A. Almeida, F. A. Musso, V. A. Carvalho, C. M. Fonseca, and G. Guizzardi, "Capturing Multi-Level Models in a Two-Level Formal Modeling Technique," in *Proc.38th ER*, 2019.

[10] C. M. Fonseca, J. P. A. Almeida, G. Guizzardi, and V. A. Carvalho, "Multi-level Conceptual Modeling: From a Formal Theory to a Well-Founded Language," in *Proc.37th ER*, 2018.

[11] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2012.

[12] F. Brasileiro, J. P. A. Almeida, V. A. Carvalho, and G. Guizzardi, "Expressive Multi-level Modeling for the Semantic Web," in *Proc.15th ISWC*, 2016.

[13] E. Mayr, *The Growth of Biological Thought: Diversity, Evolution, and Inheritance*. Harvard University Press, 1982.

[14] J. D. Lara, E. Guerra, and J. S. Cuadrado, "When and How to Use Multilevel Modelling," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, 2014.

[15] T. Kühne and D. Schreiber, "Can Programming Be Liberated from the Two-level Style: Multi-level Programming with Deepjava," in *Proc.22nd OOPSLA*, 2007.

[16] J. Odell, "Power Types," *Journal of OO Programming*, vol. 7, 1994.

[17] L. Cardelli, "Structural Subtyping and the Notion of Power Type," in *Proc.15th POPL*, 1988.

[18] T. A. Halpin, "Information Modeling and Higher-order Types," in *Proc.16th CAiSE*, 2004.

[19] G. Guizzardi, J. P. A. Almeida, N. Guarino, and V. A. de Carvalho, "Towards an Ontological Analysis of Powertypes," in *Proc.JOWO 2015*, 2015.

[20] C. Atkinson, R. Gerbig, and T. Kühne, "Comparing multi-level modeling approaches," in *Proc.1st MULTI Workshop*, 2014.

[21] F. Brasileiro, J. P. A. Almeida, V. A. Carvalho, and G. Guizzardi, "Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata," in *Proc.25th WWW*, 2016.

[22] W3C, "OWL 2 Web Ontology Language - Document Overview (Second Edition)," 2012.

[23] B. Motik, "On the Properties of Metamodeling in OWL," *Journal of Logic and Computation*, vol. 17, no. 4, 2007.

[24] K. Kimura *et al.*, "Practical Multi-level Modeling on MOF-compliant Modeling Frameworks," in *Proc.2nd MULTI Workshop*, 2015.

[25] F. Macías, A. Rutle, and V. Stolz, "MultEcore: Combining the Best of Fixed-Level and Multilevel Metamodelling," in *Proc. 3rd MULTI Workshop*, 2016.

[26] C. Atkinson and T. Kühne, "Meta-Level Independent Modelling," in *Proc.14th ECOOP*, 2000.

[27] Colin Atkinson and Thomas Kühne, "Reducing Accidental Complexity in Domain Models," *Software and System Modeling*, vol. 7, no. 3, 2008.