Toward a Well-Founded Theory for Multi-Level Conceptual Modeling

Victorio A. Carvalho[1,2] and João Paulo A. Almeida[1]

[1]Ontology & Conceptual Modeling Research Group (NEMO)
Federal University of Espírito Santo (UFES), Vitória, ES, Brazil
[2]Research Group in Applied Informatics, Informatics Department,
Federal Institute of Espírito Santo (IFES), Colatina, ES, Brazil
victorio@ifes.edu.br; jpalmeida@ieee.org

Abstract. Multi-level conceptual modeling addresses the representation of subject domains dealing explicitly with multiple classification levels. Despite the recent advances in multi-level modeling techniques, we believe that literature in multi-level conceptual modeling would benefit from a theory that: (i) formally characterizes the nature of classification levels, and (ii) precisely defines the structural relations that may occur between elements of different classification levels. This work aims to fill this gap by proposing an axiomatic theory that can be considered a reference top-level ontology for types in multi-level conceptual modeling. The theory provides the modeler with basic concepts and patterns to articulate domains that require multiple levels of classification as well as to inform the development of well-founded languages for multi-level conceptual modeling. The whole theory is founded on a basic *instantiation* relation and characterizes the concepts of *individuals* and *types*, with types organized in levels related by instantiation. Further, it includes intra-level structural relations that are used to define expressive multi-level models and cross-level relations that allow us to account for and incorporate the different notions of power type in the literature.

Keywords: Multi-level modeling, conceptual modeling, power types, clabjects, ontology.

1 Introduction

Conceptual modeling is the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication [33]. It is generally considered a fundamental activity in information systems engineering [40], in which a given subject domain is described independently of specific implementation choices [21]. The main artefact of this activity is a *conceptual model*, i.e., a specification aiming at representing a conceptualization of the subject domain of interest. Conceptual models are often used as a basis for the construction and evolution of information systems, which justifies the interest in the activity of conceptual modeling from the perspective of information systems engineering [40].

Given the scope and purpose of conceptual modeling, suitable techniques for this endeavour should be based on abstractions with consideration for human cognition and common sense [21, 19]. With this respect, there is ample psychological evidence to support the hypothesis that humans conceive of the physical and social world using some notion of "categories" and use categorization or classification strategies since a pre-language age of 3-4 months (see [21], pp. 114-118). Thus, it is no surprise that a vast majority of conceptual modeling techniques are based on notions such as "class" and "type", and that subject matter experts often refer to "kinds", "categories" and "sorts" in their accounts of a subject domain.

In several subject domains, the categorization scheme itself is part of the subject matter, and thus experts make use of categories of categories in their accounts. For instance, considering the software development domain [17], project managers often need to plan according to the *types of tasks* to be executed during the software development project (e.g. "requirements specification", "coding"). They may also need to classify those *types of tasks* giving rise to *types of types of tasks*. In this case, "requirements specification" and "coding" could be considered as examples of "technical task types", as opposed to "management task types". Finally, during project development, they need to track the execution of individual tasks (e.g. specifying the requirements of the system X). Thus, to describe the conceptualization underlying the software development domain, one needs to represent entities of different (but nonetheless related) classification levels, such as *tasks (specific individual occurrences), types of tasks,* and *types of types of tasks.* Other examples of multiple classification levels come from domains such as that of organizational roles (or professional positions) [43], biological taxonomy [31] and artefact types (e.g., product types) [36].

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 [6, 36]. 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.

The *power type* pattern [9, 39] is an example of an early approach for multi-level modeling and is used to model situations in which the instances of a type (the *power type)* are specializations of a lower-level type (the *base type).* Odell [39] illustrated the power type pattern with an example considering a type "Tree Species" having instances such as "Sugar Maple", "Apricot", "American Elm" and "Saguaro". Since all instances of "Tree Species" specialize "Tree", "Tree Species" is a power type of "Tree".

While some prominent approaches for multi-level modeling are based on the notion of *power type,* there is no consensus about the exact definition of a *power type*. For example, based on the concept of power set, Cardelli [9] coined the notion of *power type* to characterize a type that captures the common structure of *all* types that specialize a specific type (the *base type).* According to the definition of Cardelli, the *base type* itself and *any* other type whose all instances are instances of the *base type* are instances of the *power type.* In contrast, Odell defined *power type* simply as a type whose instances are subtypes of another type (the *base type*), excluding the *base type* from the set of instances of the *power type*. Differently from Cardelli, Odell admits the existence of specializations of the *base type* that are not instances of the *power type*[1].

In addition to the lack of consensus concerning the definition of *power type,* most multi-level modeling approaches based on this notion lack a formal account for it (e.g. the UML support to model power types [41] and the metamodeling framework proposed in [17]) . Further, most of

[1] This discussion is extended in this paper in section 4, where we show how the definitions are related to each other and how they can be given different uses.

these approaches represent the relation between a power type and a base type as a regular association with no specialized semantics.

Other prominent approaches for multi-level modeling (such as [5, 36]) propose to treat the instantiation between arbitrary adjacent levels uniformly [2], i.e., they defend that the instantiation relations between specific individuals and their types should also be applied to the instantiation relation occurring between types of adjacent classification levels. To meet this challenge it is necessary to admit the existence of entities, which are, simultaneously, type (class) and instance (object) [2]. The authors have coined the term "clabject" to emphasize this dual facet of classes in a generalized multi-level scheme.

Despite the recent advances in multi-level modeling techniques, we believe that the literature would benefit from a theory that: (i) formally characterizes the nature of classification levels, and (ii) precisely defines the relations that may occur between elements of different classification levels. Such a theory should be useful to guide the development of well-founded languages for multi-level conceptual modeling and to provide the modeler with basic concepts and patterns to conceptualize domains that require multiple levels of classification.

This work aims to fill this gap by proposing a theory for multi-level conceptual modeling named MLT. This theory is founded on a basic *instantiation* relation and characterizes the concepts of *individuals* and *types*, with types organized in levels related by instantiation. MLT accounts for the notion of power type with two contributions: (i) it clarifies and positions conflicting definitions of power type, and (ii) it defines new structural relations for variants of the power type pattern enriching the expressivity of multi-level modeling primitives. The basic entities in the theory and all proposed relations between entities are formally defined through axiomatization in first-order logic.

The resulting theory can be considered a reference top-level ontology for types in multi-level conceptual modeling. Although we do not propose a language for multi-level conceptual modeling, we explore patterns that emerge from the application of the theory as well as modeling constraints to ensure that multi-level models respect the theory axioms. Since our focus is on conceptual modeling (and not language engineering or language metamodeling) we focus our account on what is called "ontological instantiation" in [3] and we are thus unconcerned with "linguistic instantiation".

This paper is further organized as follows. Section 2 presents the basic entities in MLT. Section 3 discusses intra-level relations including specialization and a novel relation we call *subordination*. Section 4 discusses cross-level relations which are the basis to incorporate the notion of *power type* and its variations. Section 5 illustrates the application of the theory to the domain of biological taxonomy. Section 6 discusses the MLT accounts for attributes, relationships and dynamic classification and presents some remarks on the identity conditions of types. Section 7 discusses related work and section 8 presents conclusions and future steps.

2 MLT Foundations: Basic Types and the Instantiation Relation

The notions of *type* and *individual* are central for our multi-level modeling theory. *Types* are predicative entities that can possibly be applied to a multitude of entities (including types themselves). Particular entities, which are not types, are considered *individuals*.

Each type is characterized by an *intension, which* is used to judge whether the type applies to an entity (e.g., whether something is a Person, a Dog, a Chair) (it is also called *principle of application* in [21]). If the intension of a type *t* applies to an entity *e* then it is said that *e is an instance of t.* Thus, the *instance of* relation (or *instantiation* relation[2]) maps a type to the entities that fall under the type. The set of instances of a type is called the *extension* of the type [23]. 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 scope of the conceptualization being considered.

MLT is formalized in first-order logic, quantifying over all possible individuals and types. The *instantiation relation* is formally represented by a binary predicate *iof(e,t)* that holds if an entity *e* is instance of an entity *t* (denoting a type). For instance, the proposition *iof(Vitória,City)* denotes the fact that "Vitória" is an instance of the type "City".[3]

We build up the theory defining the conditions for entities to be considered *individuals,* with the constant "Individual" in axiom A1. An entity is an instance of "Individual" iff it does not possibly play the role of type in instantiation relations.

$$\forall \mathrm{x}\ \mathrm{iof}(\mathrm{x},\mathrm{Individual}) \leftrightarrow \nexists \mathrm{y}\ \mathrm{iof}(\mathrm{y},\mathrm{x}) \qquad (\mathrm{A1})$$

We consider that two types are equal iff the sets of all their possible instances are the same (see Axiom A2). Note that this definition of equality only applies to elements which are not *individuals*, hence the 'guard' conditions on the left-hand side of the implication.

$$\forall \mathrm{t1},\mathrm{t2}\ \big(\neg\mathrm{iof}(\mathrm{t1},\mathrm{Individual}) \land \neg\mathrm{iof}(\mathrm{t2},\mathrm{Individual})\big) \rightarrow$$
$$((\mathrm{t1}=\mathrm{t2}) \leftrightarrow (\forall \mathrm{e}\ \ \mathrm{iof}(\mathrm{e},\mathrm{t1}) \leftrightarrow \mathrm{iof}(\mathrm{e},\mathrm{t2}))) \quad (\mathrm{A2})$$

As a multi-level modeling theory, we deal with *types* that have *individuals* as instances as well as with types whose extension is composed of other types. In order to accommodate these varieties of types, the notion of *type order* is used. 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. We use the term *higher-order type* to refer to types with order higher than one.

[2] We are aware that certain approaches such as RM-ODP distinguish the terms *instantiation* and *instance*, but this distinction is not required here, and hence we use the terms interchangeably.

[3] For the sake of clarity in the presentation, we focus in this section on types that apply necessarily to their instances (the so-called rigid types [21]). A treatment of dynamic classification (and non-rigidity) is deferred to section 6.2.

Axiom A3 characterizes “First-Order-Type” (or shortly “1stOT”), defining a first-order type as an entity whose instances are instances of “Individual”. Analogously, A4 and A5 characterize “Second-Order Type” (or “2ndOT”) and “Third-Order Type” (“3rdOT”). A4 defines that an entity *t* is a second-order type iff all its instances are first-order types (i.e., instances of “1stOT”), and A5 defines that an entity *t* is a third-order type iff all its instances are second-order types (i.e., instances of “2ndOT”). This scheme can be simply extended to consider as many orders as necessary. However, since we have not encountered examples of types in conceptual modeling with order higher than three, we present our theory here for the sake of brevity considering only *first-order, second-order and third-order types*.

$$\forall t\ \mathrm{iof}(t, 1stOT) \leftrightarrow (\exists y\ \mathrm{iof}(y, t) \land (\forall x\ \mathrm{iof}(x, t) \rightarrow \mathrm{iof}(x, Individual))) \quad (A3)$$

$$\forall t\ \mathrm{iof}(t, 2ndOT) \leftrightarrow (\exists y\ \mathrm{iof}(y, t) \land (\forall t'\mathrm{iof}(t', t) \rightarrow \mathrm{iof}(t', 1stOT))) \quad (A4)$$

$$\forall t\ \mathrm{iof}(t, 3rdOT) \leftrightarrow (\exists y\ \mathrm{iof}(y, t) \land (\forall t'\mathrm{iof}(t', t) \rightarrow \mathrm{iof}(t', 2ndOT))) \quad (A5)$$

Substituting *t* for *Individual* in axiom A3 and considering A1, one can see that “Individual” is an instance of “1stOT” (theorem T1). Analogously, using further axioms A4 and A5 we can show that “1stOT” is an instance of “2ndOT” and “2ndOT” is an instance of “3rdOT” (see theorems T2 and T3).

$$\mathrm{iof}(Individual, 1stOT) \quad (T1)$$

$$\mathrm{iof}(1stOT, 2ndOT) \quad (T2)$$

$$\mathrm{iof}(2ndOT, 3rdOT) \quad (T3)$$

Theorem T4 states that “Individual”, “1stOT”, “2ndOT” and “3rdOT” have no instances in common i.e., their extensions are disjoint. To see why this theorem holds, we need to analyze all the possible combinations of the basic types in pairs, starting from evaluating the possibility for an entity to be instance of both “Individual” and “1stOT”. According to A1, instances of “Individual” do not have instances, while according to A3 instances of “1stOT” necessarily have some instance. Thus, no entity can be an instance of “Individual” and “1stOT” simultaneously. Using A1 in tandem with A4 and A5 we can conclude also that “Individual” does not have instances in common with “2ndOT” nor with “3rdOT”. Now, suppose an entity *e*, which is instance of both “1stOT” and “2ndOT”. Using A3 and A4, all its instances should be simultaneously “Individual” and “1stOT”, which is impossible, as we have already concluded. Hence, there are no entities which simultaneously instantiate “1stOT” and “2ndOT”. Following analogous reasoning and using axioms A4 and A5 one can conclude that “2ndOT” and “3rdOT” do not have instances in common. Finally, applying the same strategy and using axioms A3 in tandem with A5 one can see that “1stOT” and “3rdOT” have no entities in common.

$$\nexists x\ (iof(x, Individual) \land iof(x, 1stOT)) \lor (iof(x, Individual) \land iof(x, 2ndOT)) \lor (iof(x, Individual) \land iof(x, 3rdOT)) \lor (iof(x, 1stOT) \land iof(x, 2ndOT)) \lor (iof(x, 1stOT) \land iof(x, 3rdOT)) \lor (iof(x, 2ndOT) \land iof(x, 3rdOT)) \quad (T4)$$

Axiom A6 states that each entity in our domain of enquiry is necessarily an instance of "Individual", "1stOT", "2ndOT" or "3rdOT" (except "3rdOT" whose type is outside the scope of the formalization). This makes the set of extensions of "Individual", "1stOT", "2ndOT" and "3rdOT" a partition of the set of entities considered in the theory (and their union the domain of quantification).

$$\forall x\ (iof(x, Individual) \lor iof(x, 1stOT) \lor iof(x, 2ndOT) \lor iof(x, 3rdOT) \lor (x = 3rdOT)) \quad (A6)$$

Axioms A1 to A6 prescribe a strictly stratified organization of entities into orders. As a result, the *instance of* relation in MLT is asymmetric (i.e. irreflexive and antisymmetric) (Theorem T5) and anti-transitive (Theorem T6). These properties of instantiation relations are consistent with those widely accepted in the conceptual modeling community [23, 26].

$$\nexists x, y\ (iof(x, y) \land iof\ (y, x)) \quad (T5)$$

$$\nexists x, y, z\ (iof(x, y) \land iof(y, z) \land iof\ (x, z)) \quad (T6)$$

To see that T5 and T6 hold, one needs to observe that the stratification prescribed by axioms A1 to A6 guarantees that instantiation relations hold between two elements such that the latter is one order higher than the former. Thus, the instances of an entity are in one order lower than it, while its types are in one order higher.

To demonstrate the validity of T5 we follow a case based strategy considering all possible cases for entities in the domain of quantification according to A6:

- First, suppose *y* is an instance of "Individual". Since instances of "Individual" do not have any possible instance (A1), *iof(x, y)* is never true. Thus, T5 holds for this case.
- Suppose *y* is an instance of "1stOT". According to A3, *x* must be an instance of "Individual" to make *iof(x, y)* true. Since instances of "Individual" do not have any possible instance (A1), *iof(y, x)* is never true. Thus, T5 holds for this case.
- Suppose *y* is an instance of "2ndOT". According to A4, *x* must be an instance of "1stOT" to make *iof(x, y)* true. If *x* is instance of "1stOT", all its instances must be instances of "Individual" (A3), requiring *y* to be an instance of "Individual" to make *iof(y ,x)* true. Since *y* cannot be simultaneously instance of "2ndOT" and "Individual" (T4), T5 holds. The case in which *y* is an instance of "3rdOT" is analogous to this one.
- Finally, suppose that *y* is "3rdOT". To see why *iof(y, x)* is never true, we can consider all cases for *x* according to A6. If *x* is an instance of "Individual", *iof(y, x)* is false (A1). If *x* is an instance of "1stOT", *y* would have to be an instance of "Individual" to make *iof(y,x)* true.

However, this is not possible, as instances of "Individual" do not have any possible instance (A1), and "3rdOT" does (T3). If *x* is an instance of "2ndOT", *y* would have to be an instance of "1stOT" (A4) to make *iof(y, x)* true. Being *y* an instance of "1stOT", every instance of it would be an instance of "Individual" (A3). However, since *y* is "3rOT", its instances should be instances of "2ndOT" (A5). This is not possible, given T4. The case in which *x* is an instance of "3rdOT" is analogous. If *x* is "3rdOT", *y* would have to be instance of "3rdOT" to make *iof(y, x)* true (A5). Being *y* an instance of "3rdOT", every instance of it would be an instance of "2ndOT", which is impossible, considering that "3rdOT" and "2ndOT" have no instances in common (T4).

To demonstrate the validity of T6, we follow a case-based analysis similar to one we used to analyze T5.

- First, suppose *z* is an instance of "Individual". Since instances of "Individual" do not have any possible instance (A1), *iof(y, z)* is never true. Thus, T6 holds for this case.
- Suppose *z* is an instance of "1stOT". According to A3, *y* must be an instance of "Individual" to make *iof(y, z)* true. Since instances of "Individual" do not have any possible instance (A1), *iof(x, y)* is never true. Thus, T6 holds for this case.
- Suppose *z* is an instance of "2ndOT". According to A4, *y* must be an instance of "1stOT" to make *iof(y, z)* true. If *y* is instance of "1stOT", *x* must be an instance of "Individual" to make *iof(x, y)* true (A3). Being *z* an instance of "2ndOT" and *x* an instance of "Individual", *iof(x,z)* is never true. Thus, T6 holds for this case. The case in which *z* is an instance of "3rdOT" is analogous to this one.
- Finally, suppose that *z* is "3rdOT". In this case, to make *iof(y, z)* true, *y* must obviously be an instance of "3rdOT". If *y* is instance of "3rdOT", *x* must be an instance of "2ndOT" to make *iof(x, y)* true (A5). Being *x* an instance of "2ndOT", it cannot be instance of "3rdOT" (T4). Thus, *iof(x,z)* is never true and T6 holds in this case.

Note that the notion of order we have used is inspired on the ramified hierarchy introduced by Russell in his type theory [14]. However, Russell's main goal with the notion of order was to prevent circularity in the hierarchy of types and hence sets of a given order could include sets of an arbitrary lower order. Differently from Russell, in our theory a type can only have instances at the immediately lower order, resulting in levels of entities. This is a common feature of the *instance of* relation in various techniques which adopt the so-called *strict metamodeling* principle [2]. Further, stratified levels arise from the cascaded application of the power type pattern starting from first-order types.

Fig. 1 illustrates the elements that form the basis for our multi-level modeling theory, using a notation that is largely inspired in UML [41]. We use the UML class notation to represent the *basic types of the theory* ("Individual", "1stOT", "2ndOT" and "3rdOT"). We use associations as usual to represent relations between instances of the related types. The multiplicity of the associations reflect the constraints in the formalization. For example, each instance of "Individual" is *instance of* at least one instance of "1stOT", and, on the inverse direction, each instance of "1stOT" has at least one instance of "Individual" in its extension. We use dependencies (dashed arrows) to represent when relations hold between the types, with labels to denote the names of the predicates that apply. For instance, a dashed arrow labelled *iof* between "Individual" and "1stOT" represents that the former is an instance of the latter (i.e., that *iof(Individual,1sOT)* holds). In Fig. 1 the dashed arrows are justified by theorems T1-T3. The notation used to elaborate Fig. 1 is used in all further diagrams in this paper.

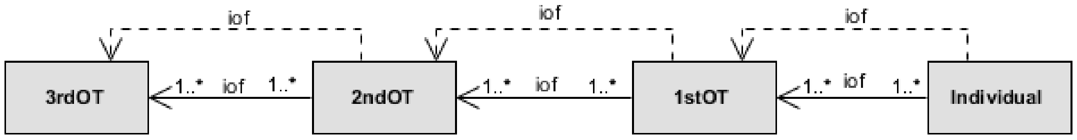

Fig. 1. Basic foundations of our multi-level modeling theory: *basic types* and *instance of* relations.

3 Intra-level structural relations

In this section, we discuss the relations that occur between types of the same order (the intra-level structural relations). All definitions are based on the instantiation relation.

We start with the ordinary specialization between types. Definition D1 defines that *t1 specializes t2* iff all instances of *t1* are also instances of *t2*. Since instances of "Individual" do not have instances (A1), D1 states that specialization only applies to elements that are not *individuals* (i.e. elements that have some possible instances). As discussed in [23, 26] *specialization* is a partial order relation (i.e., a reflexive, transitive and antisymmetric relation), which is guaranteed in MLT.

$$\forall t1, t2\ \mathrm{specializes}(t1, t2) \leftrightarrow (\exists y\ \mathrm{iof}(y, t1) \wedge (\forall e\ \mathrm{iof}(e, t1) \rightarrow \mathrm{iof}(e, t2))) \qquad \text{(D1)}$$

According to D1, every type specializes itself. Since this may be undesired in some contexts, we define the *proper specialization* relation (we used the qualifier 'proper' as in 'proper subset' considering that the extension of the specialized type is a proper subset of the extension of the general type [23]). Definition D2 thus defines that *t1 proper specializes t2* iff *t1 specializes* t2 and is different from it.

$$\forall\ t1, t2\ \ \mathrm{properSpecializes}(t1, t2) \leftrightarrow (\mathrm{specializes}(t1, t2) \wedge t1 \neq t2) \qquad \text{(D2)}$$

Insofar as the instances of a type are defined by its *intension*, the *proper specialization* relation reflects the fact that the *intension* of the specializing type keeps the constraints stated by the *intension* of the specialized type and adds some other constraint(s) to it. To put it more formally, consider two types, *t* and *t'*. If *t' proper specializes t*, this means that the intension of *t'* is given by the conjunction of the intension of *t* and a predicate that captures the additional constraints defined by *t'* with respect to *t.* Further, since we consider there is no relevant type without possible instances, the resultant intension of *t'* cannot be a trivially false predicate. For example, consider that "Man" is a type that applies to every instance of "Person" of the male gender. Assuming this, the intension of "Man" is given by the conjunction between the intension of "Person" and a predicate that captures the property of being male. Thus, in this case, "Man" *proper specializes* "Person".

Fig. 2 augments Fig. 1 by including the representation of specialization and proper specialization relations. Note that the axioms and definitions presented thus far guarantee that these relations may only hold between types of the same order, which is reflected in the diagram.

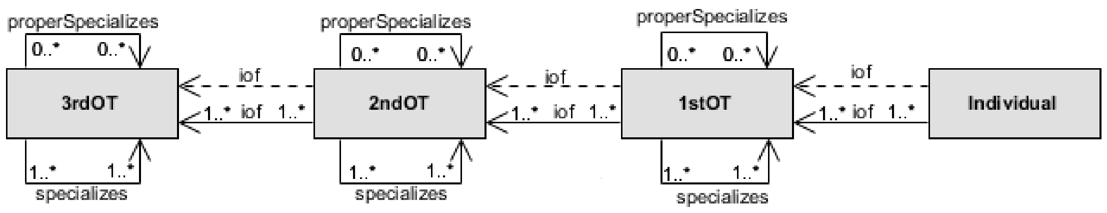

Fig. 2. Intra- level structural relations: specializations and proper specializations.

Substituting *t2* for *Individual* in definition D1 and comparing the right-hand side of the resultant proposition with the right-hand side of axiom A3, we conclude that *an entity is instance of "1stOT" iff it specializes "Individual"* (theorem T7). Analogously, it follows from D1 and A4 that *an entity is instance of "2ndOT" iff it specializes "1stOT"* (theorem T8). Finally, from D1 and A5 one can see *that every instance of "3rdOT" specializes "2ndOT"* (theorem T9). Therefore, an important consequence of the theory presented so far is that any instance of a higher-order type (any instance of "1stOT", "2ndOT", and "3rdOT") specializes the basic type at an immediately lower order.

$$\forall t\ \mathrm{iof}(t, \mathrm{1stOT}) \leftrightarrow \mathrm{specializes}(t, \mathrm{Individual}) \quad (T7)$$

$$\forall t\ \mathrm{iof}(t, \mathrm{2ndOT}) \leftrightarrow \mathrm{specializes}(t, \mathrm{1stOT}) \quad (T8)$$

$$\forall t\ \mathrm{iof}(t, \mathrm{3rdOT}) \leftrightarrow \mathrm{specializes}(t, \mathrm{2ndOT}) \quad (T9)$$

This leads to a basic pattern in the theory: Every type that is not a basic type (e.g., a domain type) is an instance of one of the basic higher-order types ("1stOT", "2ndOT", and "3rdOT"), and, at the same time specializes the basic type at the immediately lower level (respectively, "Individual", "1stOT", "2ndOT"). For example, consider the enterprise domain, in which we may need a type to capture the concept of "Employee*"*. The type "Employee" classifies *individuals* (e.g. John or Mary), i.e., every *instance of* "Employee" is also *instance of* "Individual". Thus, by axiom A3, we have that "Employee" *is instance of* "1stOT" and, considering T7, "Employee" *specializes* "Individual". In fact, since "Employee" and "Individual" are different types, we can say that “Employee” *proper specializes* “Individual”. This basic pattern is illustrated in Fig. 3. In order to preserve the intuition in the representation, we used the traditional UML notation to represent specializations (in this case to represent the fact that the proposition *properSpecializes(Employee, Individual)* holds). We have used the instance specification notation to represent an individual (John), while keeping the use of dashed arrows to show instantiation. The theory basic types are shaded to differentiate them from domain elements.

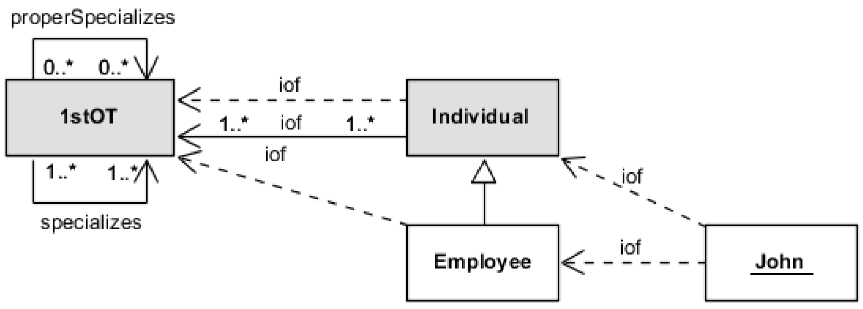

Fig. 3. Using the theory to model a domain.

MLT supports also *specializations* and *instantiations* occurring between domain elements. For instance, supposing we need to classify the employees according to their highest academic degrees we can consider types such as “PhDEmployee” and “BachelorEmployee” to classify respectively employees having Ph.D. and bachelor degrees. These types are proper specializations of “Employee” since their instances are also instances of “Employee”. Thus, by the transitivity of specialization, they also specialize “Individual” and, considering theorem T7, they are instances of “1stOT”.

Further, we may consider a *second-order* type called “EmployeeAcademicDegreeType” that has as instances the types that specialize “Employee” according to the academic degree (e.g “PhDEmployee” and “BachelorEmployee”). More formally, “EmployeeAcademicDegreeType” is a type applied to types that have the intension given by the conjunction of the intension of “Employee” and a predicate that captures the property of having a specific highest academic degree. For example, the intension of “PhDEmployee” is given by the conjunction of the intension of “Employee” and a predicate that captures the property of having a Ph.D. academic degree, thus “PhDEmployee” is instance of “EmployeeAcademicDegreeType”. Again applying the basic pattern, “EmployeeAcademicDegreeType” is an *instance of* “2ndOT” (since its instances are instances of “1stOT”) and *specializes* “1stOT” (see A4 and T8).

Fig. 4 augments Fig. 3 adding the discussed entities and relations. In order to increase the readability of the diagram, we use dashed rectangles to group elements that have a common link to another element and draw only one arrow between the border of the rectangle and the other element. For example, instead of representing two *iof* links between “EmployeeAcademicDegreeType” and its instances, we group its instances in a dashed rectangle and draw one *iof* link between such rectangle and “EmployeeAcademicDegreeType”. Moreover, we omitted the representation of some relations that are implied by the represented relations. For example, although we do not represent that “PhDEmployee” *proper specializes* “Individual” it can be inferred by the fact that it proper specializes “Employee” which, in turn, *proper specializes* “Individual”.

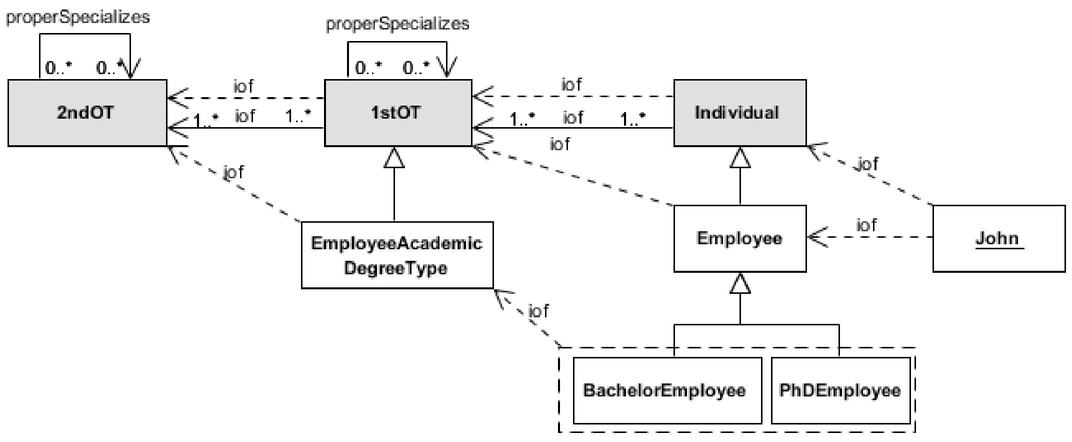

Fig. 4. Instantiations and specializations between domain elements.

Consider now an extension of the example in Fig. 4 in which we introduce a second *second-order type* called “EmployeeRoleType” beside “EmployeeAcademicDegreeType”. The instances of “EmployeeRoleType” are specializations of “Employee” according to the role they play (e.g. “Programmer” and “ResearchManager”). Consider further that, in order to reflect required qualifications in the domain, all instances of “EmployeeRoleType” must specialize instances of “EmployeeAcademicDegreeType”. In other words, the intension of each instance of “EmployeeRoleType” is given by the conjunction of the intension of an instance of “EmployeeAcademicDegreeType” and an additional constraint capturing the role their instances must play. For example, we may consider that “Programmer” *specializes* “BachelorEmployee” and “ResearchManager” *specializes* “PhDEmployee”. To allow modelers to capture this kind of relations between higher-order types that implies *specializations* between their instances MLT defines the notion of *subordination*.

We call *subordination* the relations that occur between two higher-order types *t1* and *t2* when *t1* applies to types that have the intension given by the conjunction of the intension of an instance of *t2* and a predicate that captures a constraint following some classification criteria. Therefore, D3 defines that *t1 is subordinate to t2* iff every *instance of t1* specializes an *instance of t2.* Subordination is a relation between types, and thus D3 excludes the possibility of *subordination* involving instances of “Individual” (i.e. entities with no possible instances).

$$\forall t1, t2\ \mathrm{isSubordinateTo}\ (t1, t2) \leftrightarrow$$
$$(\exists x\ \mathrm{iof}(x, t1) \land (\forall t3\ \mathrm{iof}(t3, t1) \rightarrow (\exists t4\ \mathrm{iof}(t4, t2) \land \mathrm{properSpecializes}(t3, t4)))) \qquad (D3)$$

Since *subordination* implies *specializations* between the instances of the involved types at one order lower, and *specializations* can only be established between types at the same order, *subordination* can only hold between *higher-order types* of equal order (see Fig. 5).

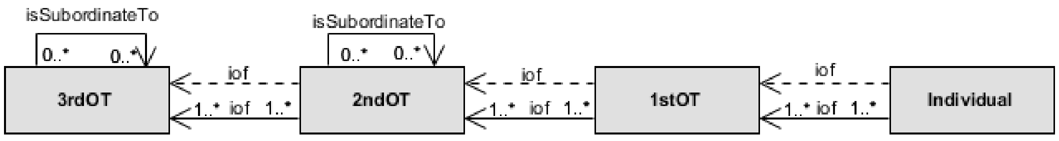

Fig. 5. Intra-level structural relations: subordination**.**

Fig. 6 illustrates the augmented example, showing that "EmployeeRoleType" *is subordinate to* "EmployeeAcademicDegreeType". Note that *subordination* between two higher-order types implies *specialization* between their instances but should be clearly distinguished from a specialization between the higher-order types (in the example, "EmployeeRoleType" does not specialize "EmployeeAcademicDegreeType"). Moreover, as we show later in section 5, the use of *subordination relations* between higher-order types plays a fundamental role on the specification of taxonomies of types in one order lower.

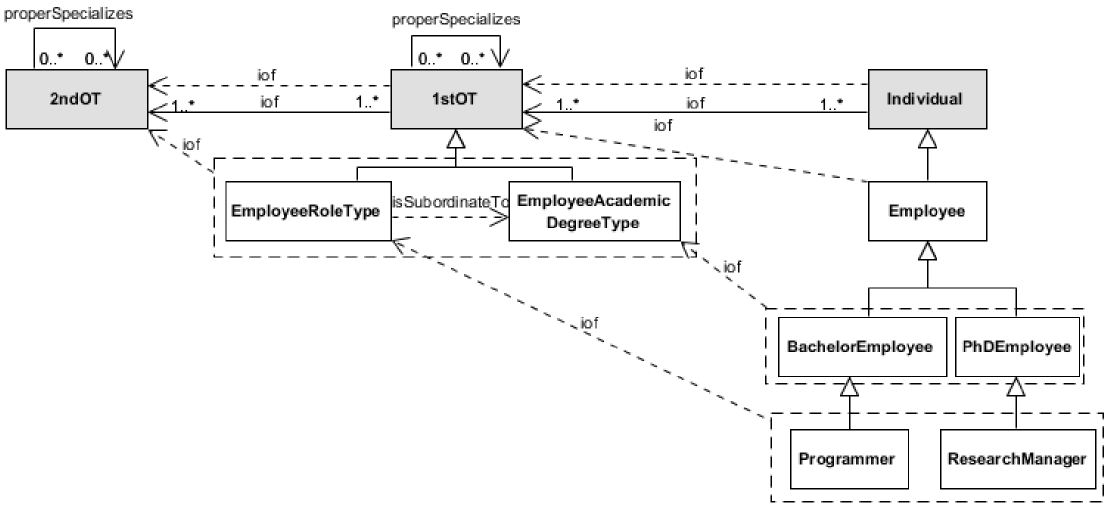

Fig. 6. An example of subordination relation.

Table 1 summarizes the characteristics of the defined intra-level structural relations.

Table 1. Intra-level structural relations characteristics

Name	Meaning	Domain and Range	Properties
Specialization *specializes(t1,t2)*	The intension of *t1* adds some classification criteria to the one of *t2* or both types have the same intension (t2=t1), i.e. every instance of *t1* is also an instance of *t2*.	Types of the same order (instances of 1stOT, 2ndOT or 3rdOT)	Reflexive, antisymmetric and transitive.
Proper Specialization *properSpecializes(t1,t2)*	The intension of *t1* adds some classification criteria to the one of *t2* i.e. every instance of *t1* is also an instance of *t2* and there at least one instance of t2 that is not instance of *t1*.		Irreflexive, antisymmetric and transitive
Subordination *isSubordinateTo(t1,t2)*	The intension of each instance of *t1* adds some classification criteria to the intension of some instance of *t2* i.e. every instance of *t1* proper specializes some instance of *t2*.	Higher-order types of the same order (instances of 2ndOT or 3rdOT)	

4 Cross-level Structural Relations

This section defines the relations that occur between types of adjacent levels (the so-called *cross-level structural relations*). These relations support our analysis of the notions of *power type* in the literature, as well as their full incorporation in the theory.

4.1 The *Power Type of* Relation

The use of *power types* is one of the most common techniques for multi-level modeling. A seminal theory for the notion of *power type* was proposed by Cardelli [9]. According to [9], the same way specializations are intuitively analogous to subsets, *power types* can be intuitively understood as powersets. The powerset of a set A, is the set whose elements are **all** possible subsets of A including the empty set and A itself. Thus, "if A is a type, then Power(A) is the type whose elements are **all** the subtypes of A" (including A) [9]. Following Cardelli's definition, definition D4 defines that iff a type *t1 is power type of* a type *t2* all *instances of t1* are *specializations of t2* and all possible *specializations of t2* are *instances of t1.* In this case, *t2* is said the base type of *t1.* Analyzing it in terms of the *intension* of the involved types, iff a type *t1 is power type of* a type *t2* the *intension of t1* defines that its instances are applicable to instances of *t2* but *t1* does not define a classification criteria. Thus, the extension of *t1* is composed by all specializations of *t2*, including *t2* itself. Further, D4 guarantees that entities without instances (*individuals*) are not considered *power types of* other entities.

$$\forall t1, t2\ \mathrm{isPowertypeOf}(t1, t2) \leftrightarrow (\exists x\ \mathrm{iof}(x, t1) \land (\forall t3\ \mathrm{iof}(t3, t1) \leftrightarrow \mathrm{specializes}(t3, t2))) \qquad (\mathrm{D4})$$

Recall that "Individual" is an *instance of* "1stOT" (theorem T1) and that all the types that specialize "Individual" are also *instances of* "1stOT" (theorem T7). Thus, it follows from the definition of *power type* (D4) that "1stOT" is *power type of* "Individual" (theorem T10). Analogously, "2ndOT" is *power type of* "1stOT" (theorem T11), and "3rdOT" is *power type of* "2ndOT" (theorem T12).

$$\mathrm{isPowertypeOf}(1stOT, Individual) \qquad (\mathrm{T10})$$

$$\mathrm{isPowertypeOf}(2ndOT, 1stOT) \qquad (\mathrm{T11})$$

$$\mathrm{isPowertypeOf}(3rdOT, 2ndOT) \qquad (\mathrm{T12})$$

It is interesting to note that, to be a *power type*, a type must have an *intension* that defines that all its instances are specializations of the *base type* and, conversely, all specializations of the *base type* are instances of the *power type* (see D4). Thus, it is possible to conclude that each type has at most one *power type* (theorem T13) and that each *type* is *power type of*, at most, one other type (theorem T14). This suggests a concrete syntactic constraint for a multi-level model: only one higher-order type can be linked to a base type through the *is power type of* relation.

$$\forall p,t\ \ \text{isPowertypeOf}(p,t) \rightarrow\ \nexists p'(p \neq p') \land \text{isPowertypeOf}(p',t) \quad \text{(T13)}$$

$$\forall p,t\ \ \text{isPowertypeOf}(p,t) \rightarrow\ \nexists t'(t \neq t') \land \text{isPowertypeOf}(p,t') \quad \text{(T14)}$$

Theorem T13 can be proved as follows: (i) supposing two higher order types, *p* and *p'*, are power type of *t*, according to D4, both *p* and *p'* should have as only instances all possible specializations of *t*; (ii) thus, applying axiom A2, we conclude that *p* is equal to *p'* (*p=p'*). Analogously, theorem T14 can be proved as follows: (i) supposing *p is power type of t*, according to D4, *p* should have as only instances all the specializations of *t*; (ii) if we also consider a type *t'* such that *p is power type of t'* then *p* should have as only instances all the specializations of *t'*; thus, *t = t'*.

In his accounts for the notion of *power type* [9], Cardelli proved that if a type *t2 specializes* a type *t1* then the *power type of t2 specializes* the *power type of t1*. Since our definition for *isPowertypeOf* relation follows Cardelli's definition, we verified that this property is entailed by our theory. Theorem T15 formalizes this property. This may be used to check the syntax of power type hierarchies, and also to generate the power type hierarchy corresponding to the base type hierarchy.

$$\forall t1,t2,t3,t4\big(\text{specializes}(t2,t1) \land \text{isPowertypeOf}(t4,t2) \land \text{isPowertypeOf}(t3,t1)\big) \rightarrow \text{specializes}(t4,t3) \quad \text{(T15)}$$

T15 can be proved as follows: (i) considering that *t3 is power type of t1* by definition D4 we conclude that *t1* and all its specializations are *instance of t3*; (ii) considering the transitivity of specialization and that *t2 specializes t1*, we have that all specializations of *t2* also *specialize t1*, and thus, all specializations of *t2* are instance of *t3*; (iii) considering that *t4 is power type of t2* by D4 we conclude that all instances of *t4* are specializations of *t2*; (iv) thus, by (ii) and (iii) we conclude that all instances of *t4* are also instances of *t3*, i.e., *t4 specializes t3*.

Given the *power type of* definition (D4), if *p1 is power type of t1* we conclude that *p1* is one order higher than *t1*, i.e., if *t1* is a *first-order* type (*iof(t1,1stOT)*) then *p1* is a second-order type (*iof(p1,2ndOT)*), if *t1* is a second-order type (*iof(t1,2ndOT)*) *p1* is a third-order type (*iof(p1,3rdOT)*), and so on. Furthermore, since instances of "Individual" are not types, they cannot participate in *isPowertypeOf* relations as *power type* nor as *base type*. Fig. 7 augments Fig. 1 by including the representation of *isPowertypeOf* relations.

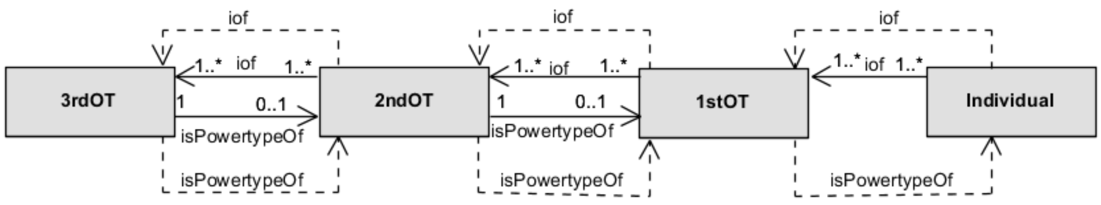

Fig. 7. Cross-level relations: *isPowertypeOf*.

Since the *power type* of a base type is a type whose intension defines that its instances classify instances of the base type, for each first-order type *f* it is always possible to define a second-order type *s* such that *s is power type of f* and for each second-order type *s* it is possible to define a third-order type *t* such that *t is power type of s*. While the theory necessitates the existence of the power type of any type (except the power types of third-order types, which are outside the scope of the theory), the decision on whether to represent the *power type* of a particular type is a modeling decision. When the *power type* is not relevant for the domain being modeled it is often omitted from the model.

To illustrate the use of the *power type of* relation, we augment the example of Fig. 6 in Fig. 8 introducing "EmployeeType", which *is power type of* "Employee". Consequently, all types that *specialize* "Employee" are *instances of* "EmployeeType". Since the instances of "EmployeeType" are first-order types, "EmployeeType" is an *instance of* "2ndOT" and *specializes* "1stOT". Further, since all instances of "EmployeeRoleType" are also *instances of* "EmployeeType", it follows that "EmployeeRoleType" *specializes* "EmployeeType". Analogously, "EmployeeAcademicDegreeType" *specializes* "EmployeeType".

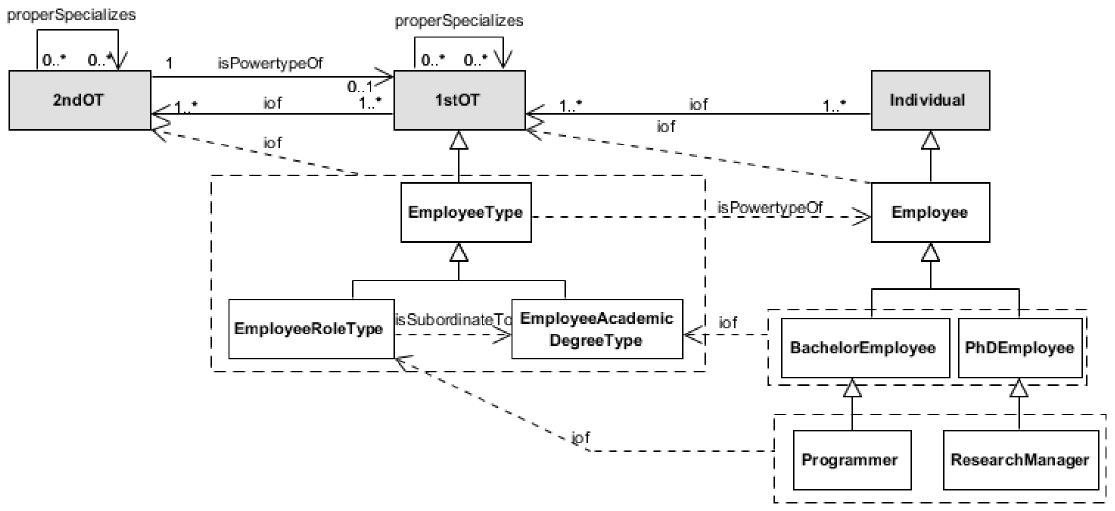

Fig. 8. An example of *isPowertypeOf* relation.

Although the definition of *power type* we adopted here is compliant with the one proposed by Cardelli [9], there are other definitions to this term in software engineering literature which have had great influence in practice, for example those definitions in [39,23].

4.2 The *Categorization* Relation

In [39], Odell stated that a *power type* is a type whose instances are subtypes of another type. It is important to notice that Odell's definition is less strict than Cardelli's [9] definition. Cardelli follows the *power set* concept stating that **all** the specializations of the base type are instances of the *power type*. Odell's definition, in turn, does not comply with that restriction. Thus, as pointed out by [23], the relation defined by Odell is misnamed *power type* since, in fact, it denotes a *subset* of the *power set*.

Inspired by Odell's definition [39] we defined the *categorization* relation (Definition D5): a type *t1 categorizes* a type *t2* iff all *instances of t1* are *properSpecializations* of *t2*. Further, D5 guarantees that *categorization relations* only apply to elements that are not *individuals* (i.e., elements that have instances).

$$\forall t1, t2 \text{ categorizes } (t1, t2) \leftrightarrow (\exists x \text{ iof}(x, t1) \land (\forall t3 \text{ iof}(t3, t1) \rightarrow \text{properSpecializes}(t3, t2))) \qquad (D5)$$

The *categorization* relation occurs between a higher order type *t1* and a base type *t2* when the *intension* of *t1* defines that their instances *specialize t2* according to a specific *classification criteria.* Thus, the instances of *t1 specialize t2* but *t2* is not an *instance of t1* and there may be other types that *specializes t2* according to other *classification criteria* and, thus, are not *instances of t1*. *Categorization* relations only occur between types of adjacent levels (see Fig. 9).

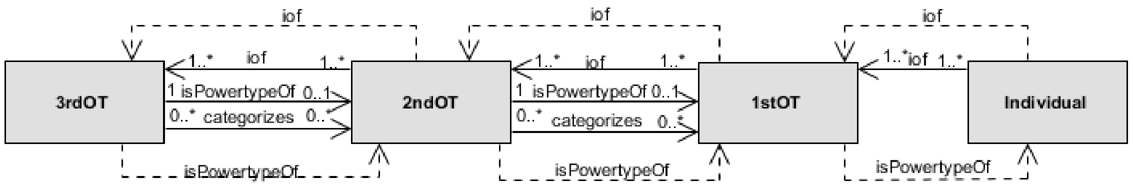

Fig. 9. Cross-Level relations: *categorizes.*

Recall that, if a type *t'* proper specializes a type *t*, the intension of *t'* is given by the conjunction of the intension of *t* and a predicate that captures the additional constraints defined by *t'* with respect to *t*. Extending this reasoning, if a higher-order type *h* categorizes *t*, the intension of *h* establishes some criteria to define the additional constraint that is introduced to the intension of *t* to compose the intension of its instances. Thus, every type *t'* whose intension extends the intension of *t* following the established criteria is considered an instance of *h*.

In our previous example, “EmployeeAcademicDegreeType” uses the employees’ academic degree as criteria to classify employees. Putting it more precisely, the intension of “EmployeeAcademicDegreeType” defines that, to be considered an instance of it, a type must have its intension given by the conjunction of the intension of “Employee” and a constraint that captures the property of having a specific academic degree. Therefore, “EmployeeAcademicDegreeType” *categorizes* “Employee”.

Still considering the previous example, the intension of “PhDEmployee” is given by the conjunction of the intension of “Employee” and a predicate that captures the property of having a PhD degree. Thus, “PhDEmployee” is an instance of “EmployeeAcademicDegreeType”. Analogously, since the instances of “EmployeeRoleType” specialize “Employee” according to roles the employees are hired to play and the academic degrees they have, “EmployeeRoleType” also *categorizes* “Employee”, having instances such as “Programmer” and “Research Manager”.

Note that, if a type *t1 is subordinate to t2* and *t2 categorizes* a type *t3*, considering the definitions of *subordination* (D3) and *categorization* (D5) we conclude that all instances of *t1 proper specialize* some instance of *t2* and that all instances of *t2 proper specialize t3.* Applying the *proper specialization* definition (D2) it follows that all instances of *t1 proper specialize t3* and, thus, *t1* *categorizes t3*. This idea is formalized in theorem T16. This theorem can be used to check the completeness of models.

$$\forall \mathrm{t1,t2,t3}\ (\mathrm{isSubordinateTo(t1,t2)} \land \mathrm{categorizes(t2,t3)}) \rightarrow \mathrm{categorizes(t1,t3)} \quad \mathrm{(T16)}$$

Further, considering the definitions of *power type* (D4), *categorization* (D5) and *proper specialization* (D2) we conclude that if a type *t2* is *power type of* a type *t1* and a type *t3 categorizes* the same base type *t1* then all instances of *t3* are also *instances of* the power type *t2* and, thus, *t3 proper specializes t2*. This idea is formalized in theorem T17. Again, this theorem can be used to check the completeness of models: a model would be incomplete if it omits the specialization between a type that categorizes a base type and this base type's power type.

$$\forall \mathrm{t1,t2,t3}\ \big(\mathrm{isPowertypeOf(t2,t1)} \land \mathrm{categorizes(t3,t1)}\big) \rightarrow \mathrm{properSpecializes(t3,t2)} \qquad \mathrm{(T17)}$$

Thus, considering our previous example, both "EmployeeAcademicDegreeType" and "EmployeeRoleType" *categorize* "Employee" and *proper specialize* "EmployeeType".

In some cases, one needs more expressiveness in the description of the relation between higher-order type and categorized type. For instance, suppose that our company considers that each employee must play at least one role, i.e., in addition to the fact that "EmployeeRoleType" *categorizes* "Employee" the instances of "EmployeeRoleType" must completely classify the instances of "Employee". In order to accommodate this expressiveness we define a variation of *categorization* relation called *complete categorization* (see definition D6). Thus, we are able to state that "EmployeeRoleType" *completely categorizes* "Employee".

$$\begin{gathered}\forall \mathrm{t1,t2}\ \mathrm{completelyCategorizes(t1,t2)} \leftrightarrow \\ (\mathrm{categorizes(t1,t2)} \land (\forall \mathrm{e}\ \mathrm{iof(e,t2)} \rightarrow \exists \mathrm{t3}\ (\mathrm{iof(e,t3)} \land \mathrm{iof(t3,t1)}))) \mathrm{(D6)}\end{gathered}$$

We also define a variation of *categorization* relation, called *disjoint categorization*, to accommodate the cases in which each *instance of* the base type is *instance of* at most one instance of the higher-order type. Thus, according to D7, a type *t1 disjointlyCategorizes t2* iff *t1 categorizes t2* and every instance of *t2* is *instance of*, at most, an *instance of t1*.

$$\begin{gathered}\forall \mathrm{t1,t2}\ \mathrm{disjointlyCategorizes}\ \mathrm{(t1,t2)} \leftrightarrow \\ (\mathrm{categorizes(t1,t2)} \land \forall \mathrm{e,t3,t4}\ ((\mathrm{iof(t3,t1)} \land \mathrm{iof(t4,t1)} \land \mathrm{iof(e,t3)} \land \mathrm{iof(e,t4)}) \rightarrow \mathrm{t3} = \mathrm{t4})) \qquad \mathrm{(D7)}\end{gathered}$$

In our example, we could consider that each employee falls under one classification according to his higher academic degree. Thus, "EmployeeAcademicDegreeType" simultaneously *disjointlyCategorizes* and *completelyCategorizes* "Employee", i.e. each *instance of* "Employee" is *instance of* one and only one *instance of* "EmployeeAcademicDegreeType". In this case we say that "EmployeeAcademicDegreeType" *partitions* "Employee" (see Fig. 10). D8 formally defines the *partitioning* relation.

$$\forall t1, t2\ \mathrm{partitions}(t1,t2) \leftrightarrow (\mathrm{completelyCategorizes}(t1,t2) \land \mathrm{disjointlyCategorizes}(t1,t2)) \quad \text{(D8)}$$

The *intension* of a higher order type which *partitions* a base type defines that its instances must apply to instances of the base type and also define a classification criteria such that each instance of the base types is classified by one and only one instance of the higher order type.

Although the definition that Odell gave to the notion of *power type* is aligned with the relation we call *categorizes*, all examples of use provided in [39] exhibit relations that should be classified as *partitions* according to our theory. Henderson-Sellers [23], following those examples of use, provided a set theoretic formalization for the notion we call here *partitions*.

Since all power type based relations (*power type of*, *categorization*, *complete categorization*, *disjoint categorization* and *partitioning*) define that the instances of their domains are *specializations/proper specializations of* their ranges, both their domains and their ranges are types. Further, their domains must be a type in one order higher than their range. Thus, only higher-order types may play the role of domain of those power type based relations. Since *complete categorization*, *disjoint categorization* and *partitioning* imply the more general *categorization* relation, only the former are represented in Fig. 10 for simplicity.

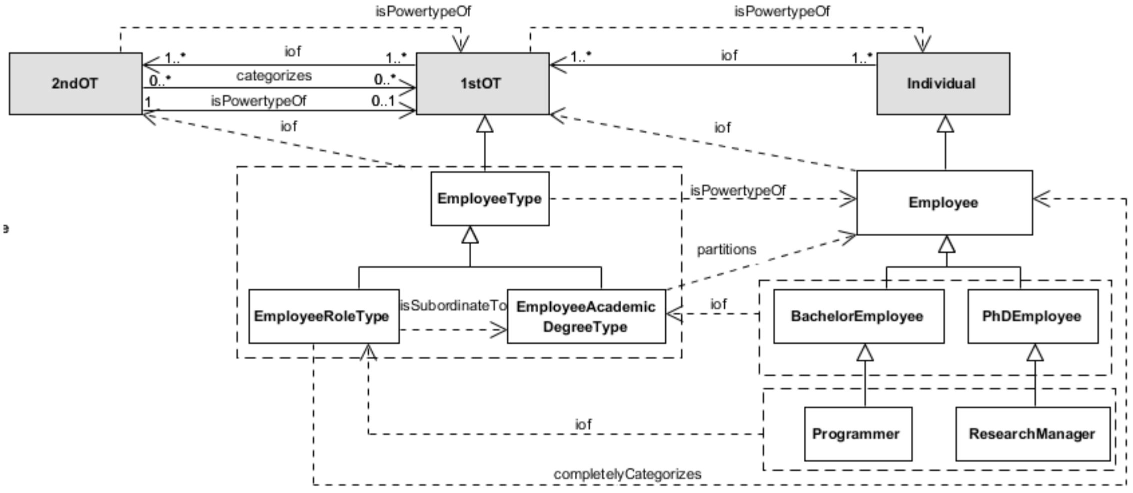

Fig. 10. An example of domain modeling applying *categorization* and *subordination* relations.

A consequence of the *partitions* definition is that, if two types *t1* and *t2* both partitions the same type *t3* then it is not possible for *t1* to specialize *t2*. This is captured in theorem T18. Again, this theorem suggests a clear syntactic constraint for a multi-level modeling language in the presence of more than one partition of the same base type.

$$\forall\ t1, t2, t3\ \left(\mathrm{partitions}(t1,t3) \land \mathrm{partitions}(t2,t3)\right) \rightarrow \neg \mathrm{properSpecializes}(t1,t2) \quad \text{(T18)}$$

T18 can be proved as follows: (i) Using the definition of *partitions* (D8), we conclude that the instances of *t1* form a disjoint and complete partition of *t3*. (ii) Supposing *t1 proper specializes t2*, using the definition of *proper specialization* (D2) we conclude that all instances of *t1* must also be instances of *t2* and *t2* must have at least one additional instance that is not an instance of *t1*. (iii) Consider that t4 is the type that is instance of t2 and is not an instance of t1. Since t2 also partitions t3, then *t4* must specialize *t3*. (iv) However, the instances of *t2* that are also instances of *t1* already completely and disjoint classify the instances of *t3*. Thus, *t4* does not have possible instances, and thus is not a valid type according to our theory. Therefore, there is no hypothesis in which *t1* partitions *t3*, *t2* partitions *t3* and *t1* specializes *t2*.

Table 2 summarizes some information about the cross-level relations. All these relations are irreflexive, antisymmetric and intransitive.

Table 2. Cross-level structural relations characteristics

Name	Meaning	Domain and Range
Instantiation *iof(e,t)*	The intension of *t* applies to *e*.	Elements of adjacent levels.
Power type of *isPowertypeOf(t1,t2)*	The intension of *t1* defines that its instances apply to instances of *t2* but do not define a *classification criteria*. Thus, the extension of *t1* is composed by all specializations of *t2*, including *t2* itself.	Types of adjacent levels (2ndOT→1stOT or 3rdOT→2ndOT)
Categorization *categorizes(t1,t2)*	The intension of *t1* defines that its instances apply to instances of *t2* according a specific *classification criteria*. Thus, the extension of *t1* is composed by the proper specializations of *t2* that follows the specified classification criteria.	
Complete Categorization *completelyCategorizes(t1,t2)*	A variation of *categorization* in which the classification criteria defined by the intension of *t1* guarantees that each instance of *t2* is instance of at least one instance of *t1*.	
Disjoint Categorization *disjointlyCategorizes(t1,t2)*	A variation of *categorization* in which the classification criteria defined by the intension of *t1* guarantees that each instance of *t2* is instance of at most one instance of *t1*.	
Partitioning *partitions(t1,t2)*	A variation of *categorization* in which the classification criteria defined by the intension of *t1* guarantees that each instance of *t2* is instance of exactly one instance of *t1*.	

5 Applying the theory to Taxonomical Structures

The previous section presented general implications of our theory for multi-level modeling. In this section we consider a representative application scenario in order to illustrate the theory expressiveness. We consider the biological taxonomy for living beings [31], which is one of the most mature examples of taxonomical hierarchies. The biological taxonomy for living beings classifies *living beings* according to *biological taxa* in seven or more ranks, e.g., *kingdom, phylum, class*, *order*, *genus*, *species*, and *breed.*

According to our theory every domain type is an instance of one of the basic higher-order types ("1stOT", "2ndOT", and "3rdOT"), and specializes the basic type at the immediately lower level (respectively, "Individual", "1stOT", "2ndOT"). Applying this pattern, we identify that (i) "LivingBeing" is an instance of "1stOT" and specializes "Individual" (since its instances are particular living beings), (ii) "BiologicalTaxon" and its specializations are instances of "2ndOT" and specializes "1stOT" (its instances are the first-order types which classify living beings, such as, e.g., the “Animalia” kingdom and the “Homo Sapiens” species[4]), and (iii) “BiologicalRank” specializes “2ndOT” and instantiates “3rdOT” (its instances are second-order types which classify taxa, such as, e.g., the “Species” taxon). Fig. 11 shows a model for this domain using the basic pattern.

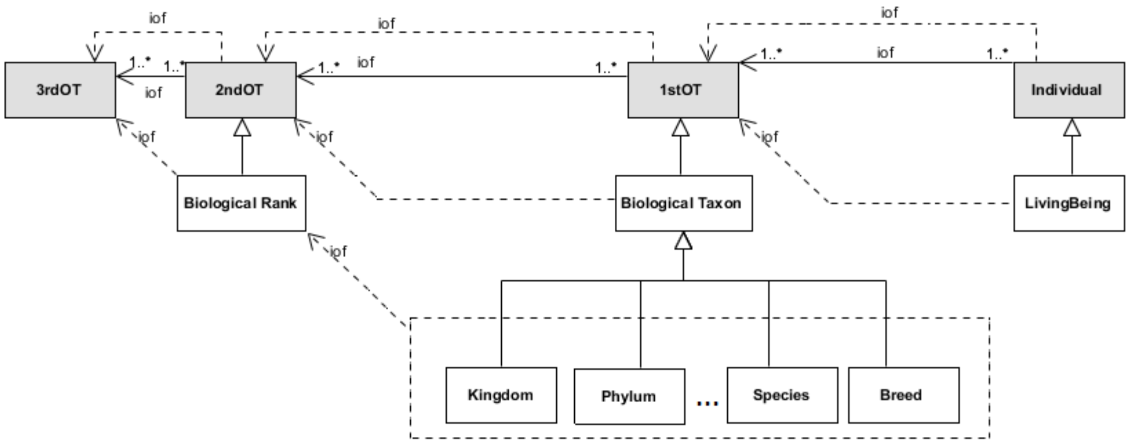

Fig. 11. Applying our theory basic pattern to the biological taxonomy for living beings.

Each “LivingBeing” is instance of one instance of each “Biological Rank”, i.e., each living being is instance of one kingdom, one phylum, and so on. Therefore, we conclude that each one of the seven instances of “BiologicalRank” *partitions* “LivingBeing”. Further, the instances of “Biological Rank” (*specializations* of “Biological Taxon”) obey a subordination chain such that every instance of “Phylum” proper specializes one instance of “Kingdom”, every instance of “Class” proper specializes one instance of “Phylum”, and so on. Thus, according to our theory, each instance of “Biological Rank” *is subordinate to* another instance of “Biological Rank”, forming a chain of subordination (except “Kingdom” which is the top of the chain). Since all instances of “Biological Rank” specialize “BiologicalTaxon” and each instance of “BiologicalTaxon” is instance of exactly one instance of “Biological Rank” (e.g., “Animal” is instance of “Kingdom”, “Collie” is instance of “Breed”, etc.) according to our theory, “Biological Rank” *partitions* “BiologicalTaxon”. Fig. 12 illustrates how the notions in the theory can be employed; one instance of each represented biological rank is shown.

[4] Note that in biology there is a long and involved debate on the ontological status of taxa such as species [15]. One of the interpretations is that biological taxa (e.g., the “Homo Sapiens” species, the “Canis Lupus Familiaris” species) represents a group of animals rather than a kind or type of animal. We stay clear of this debate and represent species (and other taxa) as the type that is instantiated by all members of that group (and only by them) (e.g., “Human” and “Dog”).

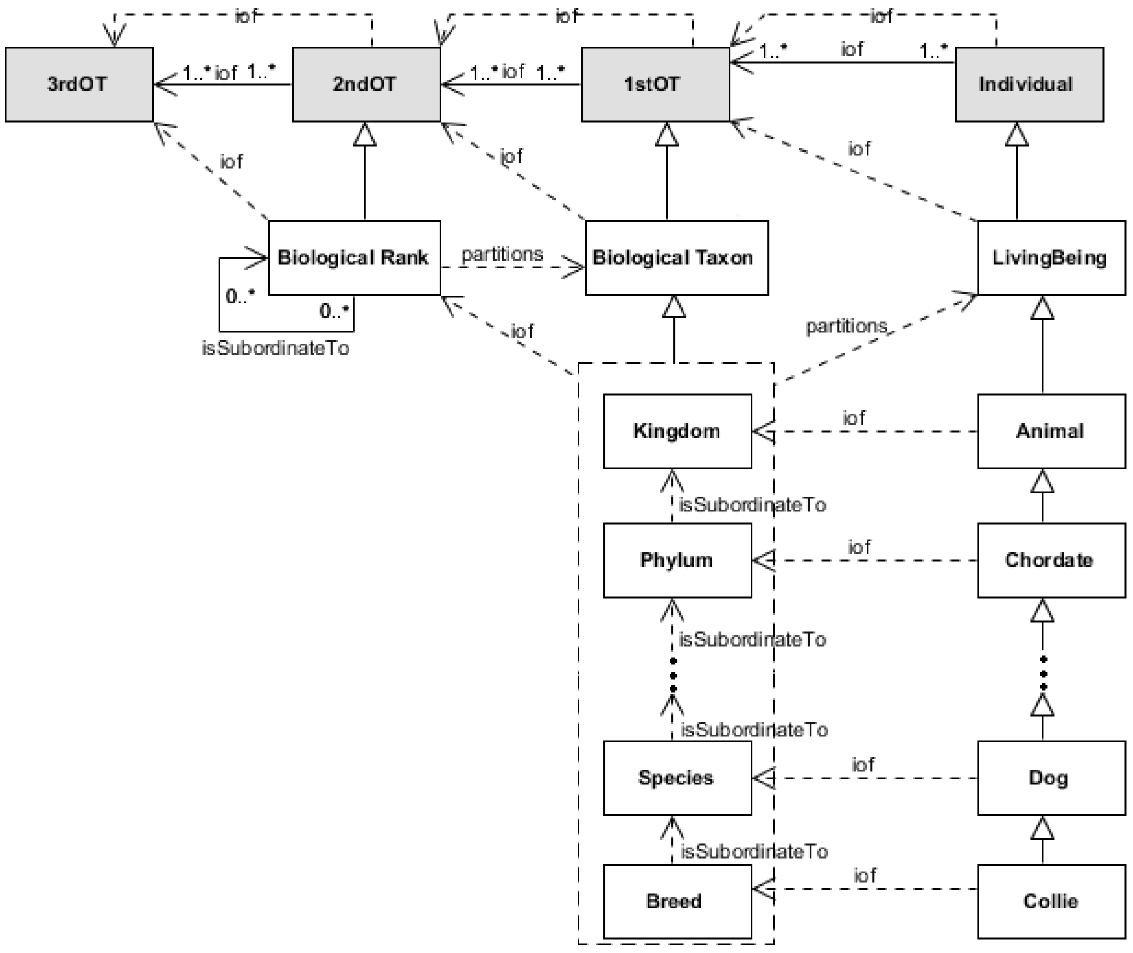

Fig. 12. Using our theory to describe the structural relations that exist in biological taxonomy (relations between the notions of biological rank, biological taxon and living being).

This example of application shows the expressiveness of our theory. We have explored the entities and relations to fully describe the structural arrangement of the biological taxonomy for living beings.

The pattern to classify domain types as instantiations and specializations of the theory basic types permitted us to identify the level of each involved concept. Using the notion of partitioning relation we were able to (i) express how the instances of biological rank apply to living beings and (ii) to understand the relation between biological rank and biological taxon. The notion of *subordination* relation was central for understanding how the instances of biological rank are related to each other.

Finally, it allowed us to notice that the shape of tree that the biological taxonomy for living beings exhibits is explained by the combination of two characteristics, namely, (i) the *partitions* relations that all instances of "BiologicalRank" have with "LivingBeing", and (ii) the chain of *subordination* that the instances of "BiologicalRank" forms.

6 Accounting for Attributes, Relationships and Dynamic Classification

6.1 Attributes and Relationships

As we have discussed so far, types capture common features of the entities that are considered their instances. If we say that "John" is an instance of the types "Person" and "Man", this is because there are certain characteristics that he shares with other instances of "Person" (such as having a brain, being a biped mammal) and with other instances of "Man" (such as having a Y chromosome). These common features are referred to in the intension of the types and are often not explicitly represented in conceptual models. Differently from these common features, features that may vary across different instances of a type or even across different points in time, are often captured using the notions of *attributes* and *relationships* (both which trace back in the conceptual modeling literature to Chen's work on the Entity Relationship model [13]). Examples of attributes are a person's height and weight, a mobile phone's screen size and a computer's storage capacity. Examples of relationships include a marriage between husband and wife, an employment between a person and an organization, the friendship between persons in a social network, etc. This section extends our account to include these ubiquitous notions in conceptual modeling.

In order to account for attributes in MLT, we extend our domain of quantification (which thus far included only *types* and *individuals*) to cover also *attributes* and their possible *values* in different possible *worlds*. In order to keep our formalization simple despite this additional sorts of elements in the domain of quantification, the axioms defined in this section are formalized in many-sorted first-order logic, assuming four disjoint sets: a set 'E' of individuals and types, a set 'A' of attributes, a set 'V' of values that can be assigned to the attributes and a set 'W' of possible worlds. The MLT axioms described in the previous sections can be understood in the light of this strategy as quantifying always over the set 'E' (composed by *individuals* and *types*).

To represent the relation between types and attributes, we define a ternary predicate *typeHasAttribute (t, a, at)* that holds if a type *t* has an attribute *a* of type *at.* For example, the proposition *typeHasAttribute (MobilePhone, serialNumber, String)* denotes that "serialNumber" is an attribute defined for the type "MobilePhone" having "String" as the type of its assignable values. Therefore, each instance of "MobilePhone" may assign instances of "String" to the attribute "serialNumber" [5].

We consider that attributes are dependent on types. To capture this notion, Axiom A7 states that for each attribute *a* there must be some entity *t* which has *a*. Further, A7 states that each attribute has a unique type *at* for its values.

$$\forall\, a{:}\,A\ (\exists\, t{:}\,E, \exists!\, at{:}\,E\ (\mathrm{typeHasAttribute}\ (t, a, at))) \qquad (A7)$$

[5] Datatypes such as *String* and *Integer* can be considered first-order types whose instances (e.g. the integer value "1" and the string "xyz") are "abstract entities" (see [21], p. 327).

To allow the representation of the values assigned to an attribute we define the predicate *hasValue(e,a,v,w)* that holds if an entity *e* assigns a value *v* to the attribute *a* in a world *w*. In order to cater for "multivalued" attributes, values assigned by entities to attributes are considered sets of entities. Therefore, the sort 'V' of possible values of attributes is, indeed, the powerset of the sort of entities 'E' (V = ℙ(E)). For instance, the proposition *hasValue(MyPhone, SerialNumber, {"1234"}, w1)* states that a specific instance of "MobilePhone", named "MyPhone", has the unitary set {"1234"} assigned to the attribute "SerialNumber" in a world "w1".

We consider that, a type *t* has an attribute *a* of type *at*, iff all instances of *t* have (at all possible worlds) a set of values *v* for *a* respecting attribute type *at* (i.e., all elements composing the set of values *v* must be instances of *at*). This definition is captured by D9. Axiom A8 defines that any entity that has a value for an attribute *a*, must be an instance of a type that has the attribute *a*. Further, we consider that the scope of an attribute is limited to a specific type and its specializations. Thus, if two different types *t* and *t'* have a common attribute *a* it means that there is a type *t''* such that *t''* has the attribute *a* and both *t* and *t' specializes t''* (see axiom A9).

$$\begin{gathered}\forall t{:}\,E, a{:}\,A\ (\mathrm{typeHasAttribute}\ (t,a,at) \leftrightarrow \\ (\neg \mathrm{iof}(t,\mathrm{Individual}) \wedge \neg \mathrm{iof}(at,\mathrm{Individual}) \wedge \forall e{:}\,E(\ \mathrm{iof}(e,t) \rightarrow \\ \forall w{:}\,W, \exists!\, v{:}\,V\ \Big(\mathrm{hasValue}(e,a,v,w) \wedge\ \forall e'{:}\,E\big(e' \in\ v \rightarrow \mathrm{iof}(e',at)\big)\Big)))) \qquad \text{(D9)}\end{gathered}$$

$$\forall e{:}\,E, a{:}\,A, v{:}\,V, w{:}\,W\ (\mathrm{hasValue}(e,a,v,w) \rightarrow \exists\ t, at : E\ (\mathrm{iof}(e,t) \wedge\ \mathrm{typeHasAttribute}\ (t,a,at))) \qquad \text{(A8)}$$

$$\begin{gathered}\forall t,t',at{:}\,E, a{:}\,A\ ((\mathrm{typeHasAttribute}\ (t,a,at) \wedge \mathrm{typeHasAttribute}\ (t',a,at)) \rightarrow \\ \exists\ t''{:}\,E\ (\ \mathrm{typeHasAttribute}\ (t'',a,at) \wedge \mathrm{specializes}(t,t'') \wedge\ \mathrm{specializes}(t',t''))) \qquad \text{(A9)}\end{gathered}$$

As a consequence of the definitions and axioms present so far, if a type *t' specializes t*, then *t'* has all attributes of *t*, capturing the semantics of inheritance (see theorem T19). Theorem T19 can be proved as follows: (i) considering that *t* defines an attribute *a*, by A9 we infer that all instances of *t* must assign values to *a*; (ii) Since *t'* specializes *t*, by the specialization definition we conclude that all instances of *t'* are also instances of *t*, and thus, all instances of *t'* must assign values to *a;* (iii) Therefore, by D9 we conclude that *t'* also has the attribute *a*. Another consequence that follows from the definitions and axioms defined so far is that given an attribute *a* there exists one topmost type *t* that defines *a*, i.e. there is a type *t* that has *a* such that any other type *t'* that has the attribute *a specializes t* (see T20).

$$\forall t,t',at{:}\,E, a{:}\,A\ (\ (\mathrm{typeHasAttribute}\ (t,a,at) \wedge \mathrm{specializes}\ (t',t)) \rightarrow \mathrm{typeHasAttribute}\ (t',a,at)) \qquad \text{(T19)}$$

$$\forall\ a{:}\,A, \exists\ !\ t, at{:}\,T\ (\mathrm{typeHasAttribute}\ (t,a,at) \wedge \forall\ t'{:}\,T\big(\ \mathrm{typeHasAttribute}(t',a,at) \rightarrow \mathrm{specializes}(t',t)\big)) \qquad \text{(T20)}$$

The use of sets as values to the attributes allows the representation of multivalued attributes (by setting as the attribute value a set with more than one element) and the representation of optional attributes by allowing attributes to have a null set as value. Definitions D10 and D11 capture the notions of mandatory and monovalued attributes in order to express constraints on the multiplicities of attributes. An attribute *a* is mandatory iff in every possible world, the values assigned to it are not empty sets. An attribute *a* is monovalued iff in every possible world, the values assigned to it by all entities are sets containing at most one value. Therefore, in order to express, for example, that each instance of "MobilePhone" has one and only one "serialNumber", besides defining that *typeHasAttribute (MobilePhone, SerialNumber, String)* one should also state that *isMandatoryAttribute(SerialNumber)* and *isMonoValuedAttribute(SerialNumber).*

$$\forall \mathrm{a}{:}\,\mathrm{A}\ (\mathrm{isMandatoryAttribute}\ (\mathrm{a}) \leftrightarrow \forall \mathrm{e}{:}\,\mathrm{E}, \mathrm{v}{:}\,\mathrm{V}, \mathrm{w}{:}\,\mathrm{W}(\mathrm{hasValue}(\mathrm{e},\mathrm{a},\mathrm{v},\mathrm{w}) \rightarrow \exists \mathrm{e}'{:}\,\mathrm{E}(\mathrm{e}' \in \mathrm{v}))) \qquad (\mathrm{D}10)$$

$$\forall \mathrm{a}{:}\,\mathrm{A}\ (\mathrm{isMonoValuedAttribute}\ (\mathrm{a}) \leftrightarrow \\ \forall \mathrm{e}{:}\,\mathrm{E}, \mathrm{v}{:}\,\mathrm{V}, \mathrm{w}{:}\,\mathrm{W}(\mathrm{hasValue}(\mathrm{e},\mathrm{a},\mathrm{v},\mathrm{w}) \rightarrow \forall \mathrm{e}', \mathrm{e}''{:}\,\mathrm{E}((\mathrm{e}' \in \mathrm{v} \land \mathrm{e}'' \in \mathrm{v}) \rightarrow \mathrm{e}' = \mathrm{e}''))) \ (\mathrm{D}11)$$

In the case of multi-level modeling, attributes defined in higher-order types can be given a value for types. We assume that attributes defined in one order capture properties of elements of the immediately lower order and, thus, may have values assigned to them in one order lower. In other words, attributes defined in first-order types have values assigned for individuals, attributes defined in second-order types have values assigned for first-order types, and so on.

Fig. 13 illustrates the concepts presented so far. To capture that each instance of "MobilePhone" must have an IMEI number, a screen of a specific size and a specific storage capacity the "MobilePhone" type defines three *mandatory* and *monovalued* attributes, namely "imei", "screenSize" and "storageCapacity". Therefore, assuming that all these attributes have values of type "String" we may state that *typeHasAttribute(MobilePhone, Imei, String), typeHasAttribute(MobilePhone, ScreenSize, String),* and *typeHasAttribute(MobilePhone, StorageCapacity, String)* hold.

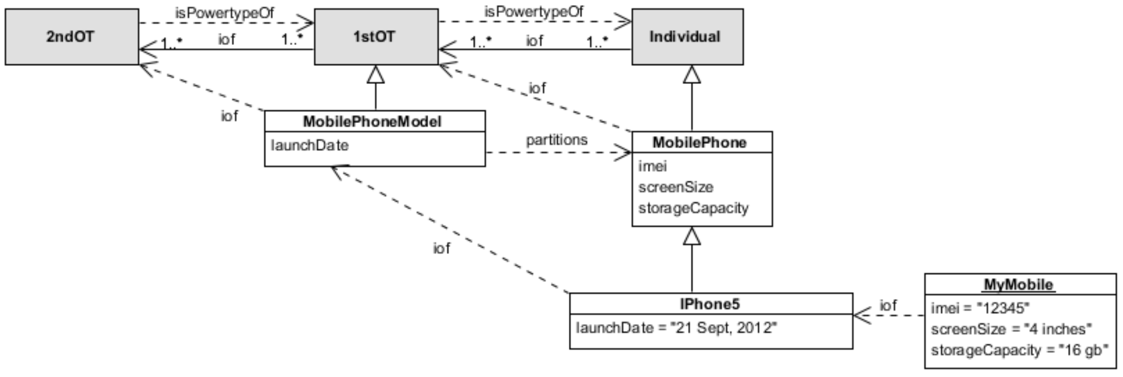

Fig. 13. Illustrating the account for attributes.

In Fig. 13, "MyMobile" is an instance of "MobilePhone" (i.e. *iof(MyMobile, MobilePhone)* holds) having "12345" as its IMEI number, a "4-inch" screen and "16 GB" of storage capacity (in Fig. 13 we represented the assignment of values to attributes by adding, after the attribute name, an equality "=" followed by the valued assigned to the attribute; attributes are considered by default mandatory and monovalued). Assuming that Fig. 13 illustrates the state-of-affairs of a world *w1*, we may state that *hasValue(MyMobile, Imei,{"12345"}, w1)*, *hasValue(MyMobile, ScreenSize,{"4 inches"}, w1)* and *hasValue(MyMobile, StorageCapacity,{"16 GB"}, w1)* hold.

Further, considering that each instance of "MobilePhone" must be classified by one instance of "MobilePhoneModel", we define a type "MobilePhoneModel" that *partitions* "MobilePhone" (i.e., *partitions(MobilePhoneModel, MobilePhone)* holds). To capture the official launch date of each mobile phone model, we define that "MobilePhoneModel" has an attribute named "launchDate" (*typeHasAttribute(MobilePhoneModel, LaunchDate, String)*). In Fig. 13 "IPhone5" is an instance of "MobilePhoneModel" launched on "21 Sept., 2012" (i.e. *hasValue(IPhone5, LaunchDate, {"21 Sept., 2012"}, w1)* holds).

All attributes introduced in the example so far only have effects at the immediately lower level, complying thus to what has been called "shallow instantiation" [6]. However, a key characteristic of an account for attributes in a multi-level theory is that attributes defined in higher-order types (such as second- and third-order types) may affect the intension of the instances of these 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. Following [22] we classify these attributes as *regularity attributes*.

Definition D12 formalizes the notion of *regularity attributes* as attributes that affect the intension of the instances of the types that have it, i.e. two instances having different values assigned to a regularity attribute must have different instances. Therefore, recalling that in MLT two types are the same if they have the exact same possible instances, D12 defines that, an attribute *a* is a regularity attribute iff every different value for *a* results in a different type[6]. Further, since regularity attributes affect the intension of instances of a type, they are necessarily defined by *higher-order types*. This constraint is reflected in D12.

$$\begin{gathered}\forall a{:}\,A\ (\mathrm{regularityAttribute}\ (a) \leftrightarrow \\ (\forall t, at{:}\,E\ (\mathrm{typeHasAttribute}\ (t, a, at) \rightarrow (\mathrm{iof}(t, 2ndOT) \vee \mathrm{iof}(t, 3rdOT))) \wedge \\ \forall t, t'{:}\,E, v, v'{:}\,V, w{:}\,W\ ((\mathrm{hasValue}(t, a, v, w) \wedge \mathrm{hasValue}(t', a, v', w) \wedge v \neq v') \rightarrow t \neq t')))\end{gathered} \qquad (D12)$$

Fig. 14 extends Fig. 13 adding to "MobilePhoneModel" the attributes "instancesScreenSize", "instancesMinStorageCapacity" and "instancesMaxStorageCapacity". All these attributes effectively serve as parameters in the intension of the instances of "MobilePhoneModel", i.e. the values assigned to these attributes influence the selection of the possible instances of instances of "MobilePhoneModel". Therefore, they are considered *regularity attributes*. For example, by assigning the value "4 inches" to the attribute "instancesScreenSize" of "IPhone5" we are representing that every instance of "IPhone5" must have 4-inch screens. Analogously, by assigning the values "16

[6] A more comprehensive definition would acknowledge that differences in various regularity attributes simultaneously may cancel each other's effects on the intension, thus we could add a *ceteris paribus* clause to definition D12, which would then state that an attribute *a* is a regularity attribute iff different values for *a* with *all other things equal* would result in a different type.

GB" and "32 GB" respectively to the attributes "instancesMinStorageCapacity" and "instancesMaxStorageCapacity" of "IPhone5" we are representing that every instance of "IPhone5" must have storage capacity between 16 and 32 GB. Therefore, having a 4-inch screen and storage capacity between 16 and 32 GB are parts of the intension of "IPhone5".

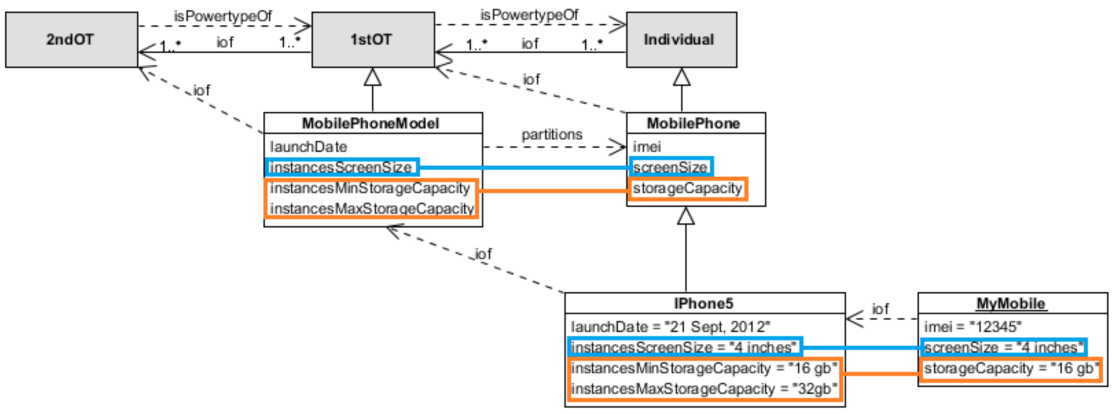

Fig. 14. Illustrating the notion of *regularity attributes*.

The influence of the regularity attributes of higher-order types over the intension of its instances may be reflected as constraints over the possible values for attributes of the base type. For example, the fact that "instancesScreenSize" is a *regularity attribute* of "MobilePhoneModel" is reflected by the fact that an instance of "MobilePhoneModel" must have as instances mobile phones having a specific "screenSize". Therefore, the fact that "IPhone5" has the value "4 inches" assigned to the *regularity attribute* "instancesScreenSize" implies that every instance of "IPhone5" must have the value "4 inches" assigned to the attribute "screenSize". Analogously, the values assigned to the *regularity attributes* "instancesMinStorageCapacity" and "instancesMaxStorageCapacity" of "MobilePhoneModel" constrain the possible values the instances of a specific (instance of) "MobilePhoneModel" may have. For example, the values "16GB" and "32 GB" respectively assigned to "instancesMinStorageCapacity" and "instancesMaxStorageCapacity" of "IPhone5" imply that its instances must have a value between 16 and 32 GB assigned to the "storageCapacity" attribute. To emphasize the relations between the *regularity attributes* of "MobilePhoneModel" and the attributes of "MobilePhone" and the constraints over their values, in Fig. 14 we placed the related attributes in colored boxes that are linked to each other. Note that this is not meant as a modeling language construct, and our sole intention here is to draw attention to these relations involving regularity attributes.

A mechanism to express the relations between regularity attributes defined by types in one order and attributes of types in one order lower is a desirable feature of multi-level modeling languages. Indeed, some potency-based approaches to multi-level modeling include some support for the representation of regularity attributes. For example, in Melanie [1] the notions of *durability* and *mutability* are used to capture situations in which the regularity attribute of a higher-order type *t* directly influences the possible values that instances of instances of *t* may assign to an attribute (such as the relation between the attribute “instancesScreenSize” of “MobilePhone-Model” and the attribute “screenSize” of “MobilePhone” illustrated in Fig. 14). For further discussion considering the relation between MLT and clabject and deep-instantiation based approaches see the related work section.

To account for basic relations in our theory we follow a strategy similar to the one adopted by OWL [49], Telos [34] and Ecore [47]: we represent a binary relation between two types *t1* and *t2* as an attribute defined in *t1* with type *t2* (actually representing an association end connected to *t2*). This treatment allows the reuse of the notions of mandatory, monovalued and regularity attributes. These can be applied on both association ends when necessary, giving rise to two opposing attributes each one defined in each of the related types having the other type as the type of the attribute (again similarly to Ecore and OWL). For example, consider the scenario illustrated in Fig. 15. According to Fig. 15, each instance of “MobilePhone” must have an instance of “Processor” installed in it (“installedProcessor”) and each instance of “Processor” may be “installed in”, at most, one instance of “MobilePhone”. Thus: (i) “MobilePhone” has a mandatory and mono-valued attribute called “installedProcessor” having “Processor” as type (formally, *typeHasAtribute(MobilePhone, installedProcessor, Processor), isMandatoryAttribute(installedProcessor)* and *isMonoValuedAttribute(installedProcessor)*); and (ii) “Processor” has a mono-valued attribute called “installedIn” having “MobilePhone” as type (formally, *typeHasAtribute(Processor, installedIn, MobilePhone)* and *isMonoValuedAttribute(installedIn)*).

Following the same approach, we could represent the relation between “MobilePhoneModel” and “ProcessorModel” depicted in Fig. 15. To capture that each (instance of) “MobilePhone-Model” is compatible with one (instance of) “ProcessorModel” we could define in “MobilePhone-Model” a mandatory mono-valued attribute called “compatibleProcessorModel” having “ProcessorModel” as type (formally, *typeHasAtribute(MobilePhoneModel, compatibleProcessorModel, ProcessorModel)*). Moreover, we could capture the fact that each (instance of) “ProcessorModel” may be compatible with some (instance of) “MobilePhoneModel” by defining in “Processor-Model” an attribute called “compatiblePhoneModels” having “MobilePhoneModel” as type (formally, *typeHasAtribute(ProcessorModel, compatiblePhoneModels, MobilePhoneModel).*

Whenever opposite attributes are defined, we need to relate the two attributes in order to constrain that whenever an instance of one type refers to an instance of the other type through an attribute, the reference in the opposite direction also holds. For example, we need to constrain that whenever an instance *e* of “MobilePhone” *refers to* an instance *e’* of “Processor” through the attribute “installedProcessor” then *e’ refers to e* through the attribute “installedIn”. In this case, we say that the attributes “installedProcessor” and “installedIn” *are opposite* attributes. The *refers to* predicate is formally defined in D13, while D14 formally defines the *is opposite* predicate. D13 states that an entity *e* refers to an entity *e’* through attribute *a* (in a world *w*), iff the value of *e* for *a* (in *w*) is a set that includes *e’*. D14, in its turn, states that two attributes *a* and *a’ are opposite* iff whenever an entity *e refers to* an entity *e'* through the attribute *a, e' refers to e* through a' and vice versa. In the example below, *isOpposite(installedIn, installedProcessor) and isOpposite(compatiblePhoneModels, compatibleProcessorModel)* hold.

$$\forall e, e': E, a: A, w: W\ (\mathrm{refersTo}(e, e', a, w) \leftrightarrow \forall\ v: V(\mathrm{hasValue}(e, a, v, w) \rightarrow (e' \in\ v))) \qquad (D13)$$

$$\forall a, a': A\ (\mathrm{isOpposite}\ (a, a') \leftrightarrow\ \forall e, e': E, w: W\ (\mathrm{refersTo}(e, e', a, w) \leftrightarrow\ \mathrm{refersTo}(e', e, a', w))) \qquad (D14)$$

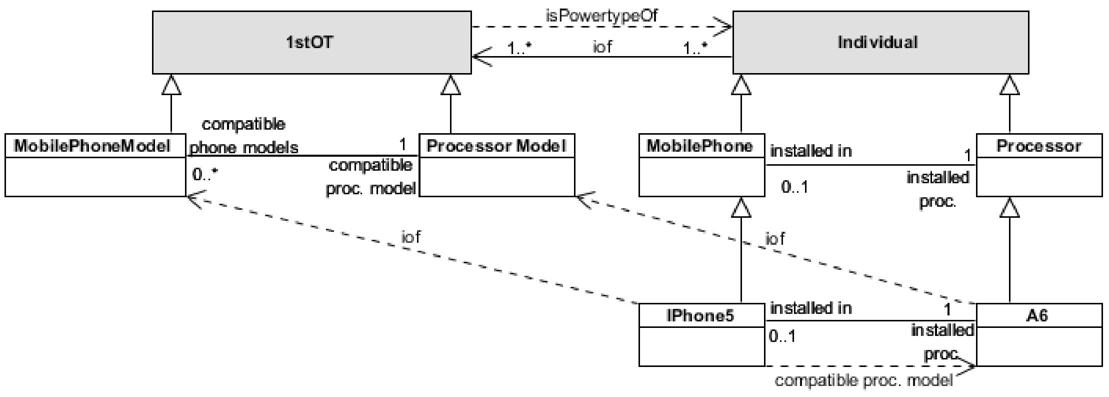

Fig. 15. Illustrating a scenario in which relations in one order capture regularities over instances of types in one order lower.

To see how the notion of *regularity attribute* is applicable to the attributes capturing relations between types, consider that instances of a (instance of) "MobilePhoneModel" may have installed on them only instances of the (instance of) "ProcessorModel" compatible with it. In this case, the intension of an instance of "MobilePhoneModel" is affected by the value assigned to its "compatibleProcessorModel" attribute. For example, in Fig. 15 since the "IPhone5" has "A6" as "compatibleProcessorModel" instances of "IPhone5" must have processors of type "A6" installed on them. Therefore, the attribute "compatibleProcessorModel" of the type "MobilePhoneModel" is a *regularity attribute* that constrains the possible values for the attribute "installedProcessor" of the type "MobilePhone". Conversely, and following analogous reasoning, we can conclude that the attribute "compatiblePhoneModel" of the type "ProcessorModel" is a *regularity attribute* that constrains the possible values for the attribute "installedIn" of the type "Processor".

This simple treatment of relations as attributes can be extended with a notion of relations as object-like entities [20]. This notion was already discussed by Chen in 1976 [13], where he observes that "some people may view something (e.g. a marriage) as a relationship" (i.e. as a tuple that relates two entities), "while other people may view it as an entity" (i.e. as something that have its own life). Subscribing Chen's intuition and aiming to explain the very nature of relationships, Guarino and Guizzardi presented in [20] an ontological theory of relationships as object-like entities. Following such theory, the relations can be reified giving rise to the so-called relator types.

Following [20], MLT supports the representation of relator types as regular types. The formalization that we propose for attributes can also be used to establish the link between relata and relators. Further, the relator-based approach [20] can be used to address n-ary relationships when necessary. An example of the use of relator types in multi-level modeling with MLT can be seen in [10] (including examples of types of relator types in the organizational structure domain).

6.2 Dynamic Classification

The MLT formalization in section 2 assumed for simplification that entities instantiate types necessarily, effectively dealing with rigid types in a static classification setting. In this section, we lift that restriction and discuss how dynamic classification can be addressed in MLT, accounting thus also for non-rigid types. Dynamic classification is key to conceptual modeling and in particular to ontology-based conceptual modeling [21]. Supporting dynamic classification allows the use of MLT as a basis for these kinds of conceptual models (e.g. see [12] and [10]).

By addressing dynamic classification, we want to support the notion that both individuals and types can change qualitatively keeping their identity. Consider for example, a hierarchy of second-order types in which the second-order type "Species" is specialized according to conservation status into "Not Threatened", "Endangered" and "Extinct". Making the types "Not Threatened", "Endangered" and "Extinct" anti-rigid allows us to capture the fact that a particular species (say "Giant Panda") can change types. Ontological implications of this approach to the nature of types are discussed in [22].

Our strategy to formalize this notion is based on the use of a world-indexed *instance of* relation, represented by a ternary predicate *iof(e,t,w)* that holds if an entity *e* is instance of an entity *t* (denoting a type) in a world *w*. Consider for example that "John" is an instance of "Student" at world "w1" but not at "w2", when he has graduated. In this case, we can state that *iof(John,Student,w1)* and ¬*iof(John,Student,w2).*

This modification to the instantiation predicate requires us to adjust some axioms of MLT accordingly. Axiom A1 was modified to express that, to be considered an instance of "Individual", an entity must have no possible instance in any admissible world (see axiom A1'). Further, two types are considered the same iff they have the same instances in all possible worlds (see axiom A2'). Thus, two types whose extensions are contingently equal are not considered the same. For example, it allows us to capture that, although there is a possible world in which all instances of "Person" are also instances of "Student", "Student" and "Person" are different types since an (instance of) "Person" is not necessarily an (instance of) "Student".

$$\forall x, w\ (\mathrm{iof}(x, \mathrm{Individual}, w) \leftrightarrow \forall w'(\mathrm{world}(w') \rightarrow \neg\exists y\ (\mathrm{iof}(y, x, w')))) \qquad \text{(A1')}$$

$$\begin{gathered}\forall t, t', w\ ((\neg\mathrm{iof}(t, \mathrm{Individual}, w) \wedge \neg\mathrm{iof}(t', \mathrm{Individual}, w)) \rightarrow \\ ((t = t') \leftrightarrow \forall x, w'(\mathrm{iof}(x, t, w') \leftrightarrow \mathrm{iof}\ (x, t', w')))) \qquad \text{(A2')}\end{gathered}$$

The characterizations of the other basic types must also be adjusted to consider possible worlds. Thus, axiom A3' characterizes "First-Order-Type" (or shortly "1stOT"), defining a first-order type as an entity with at least one instance in a possible world and whose instances in all possible worlds are instances of “Individual”. Analogously, A4’ and A5’ characterize “Second-Order Type” (or “2ndOT”) and “Third-Order Type” (“3rdOT”). Additionally, axiom A6’ adjusts A6 to state that, for *all possible worlds*, each entity in our domain of enquiry is either an instance of “Individual”, “1stOT”, “2ndOT” or “3rdOT” (except “3rdOT” whose type is outside the scope of the formalization).

$$\forall \mathrm{t}, \mathrm{w}\ (\mathrm{iof}(\mathrm{t}, 1\mathrm{stOT}, \mathrm{w}) \leftrightarrow (\exists \mathrm{y}, \mathrm{w}'\ (\mathrm{iof}(\mathrm{y}, \mathrm{t}, \mathrm{w}')) \wedge \forall \mathrm{x}, \mathrm{w}''\ (\mathrm{iof}(\mathrm{x}, \mathrm{t}, \mathrm{w}'') \rightarrow \mathrm{iof}(\mathrm{x}, \mathrm{Individual}, \mathrm{w}'')))) \quad \text{(A3’)}$$

$$\forall \mathrm{t}, \mathrm{w}\ (\mathrm{iof}(\mathrm{t}, 2\mathrm{ndOT}, \mathrm{w}) \leftrightarrow (\exists \mathrm{y}, \mathrm{w}'\ (\mathrm{iof}(\mathrm{y}, \mathrm{t}, \mathrm{w}')) \wedge \forall \mathrm{t}', \mathrm{w}''\ (\mathrm{iof}(\mathrm{t}', \mathrm{t}, \mathrm{w}'') \rightarrow \mathrm{iof}(\mathrm{t}', 1\mathrm{stOT}, \mathrm{w}'')))) \quad \text{(A4’)}$$

$$\forall \mathrm{t}, \mathrm{w}\ (\mathrm{iof}(\mathrm{t}, 3\mathrm{rdOT}, \mathrm{w}) \leftrightarrow (\exists \mathrm{y}, \mathrm{w}'\ (\mathrm{iof}(\mathrm{y}, \mathrm{t}, \mathrm{w}') \wedge \forall \mathrm{t}', \mathrm{w}''\ (\mathrm{iof}(\mathrm{t}', \mathrm{t}, \mathrm{w}'') \rightarrow \mathrm{iof}(\mathrm{t}', 2\mathrm{ndOT}, \mathrm{w}'')))) \quad \text{(A5’)}$$

$$\begin{gathered}\forall \mathrm{w}, \mathrm{x}\ ((\mathrm{world}(\mathrm{w}) \wedge \neg \mathrm{world}(\mathrm{x})) \rightarrow \\ (\mathrm{iof}(\mathrm{x}, \mathrm{Individual}, \mathrm{w}) \vee \mathrm{iof}(\mathrm{x}, 1\mathrm{stOT}, \mathrm{w}) \vee \mathrm{iof}(\mathrm{x}, 2\mathrm{ndOT}, \mathrm{w}) \vee \mathrm{iof}(\mathrm{x}, 3\mathrm{rdOT}, \mathrm{w}) \vee (\mathrm{x} = 3\mathrm{rdOT}))) \quad \text{(A6’)}\end{gathered}$$

Since axiom A1’ defines that to be an instance of “Individual” in a world *w* an entity *x* must not have instances in any world, we can conclude that if an entity *x* is an instance of “Individual” in any world it is an instance of “Individual” in all possible worlds, i.e. instances of “Individual” are necessarily instances of it. Thus, “Individual” is a rigid type. Analogously, using axioms A3’ to A5’ we conclude that “1stOT”, “2ndOT” and “ 3rdOT” are also rigid types, i.e., the basic types of MLT are all rigid.

Concerning the intra- and the cross-level structural relations of MLT, they express properties that are not contingent to the involved types. Consider for example the specialization relation between *t1* and *t2*: a type *t1* specializes a type *t2* iff in all possible worlds all instances of *t1* are also instances of *t2* (see definition D1’). We can observe that, by definition, it is not admissible for *t1* to *specialize t2* in a world *w* and not specialize it in another world *w’*. Thus, in contrast with the instantiation relation, the *specialization* relation is not world-indexed. The same reasoning applies to all other intra- and cross-level structural relations of MLT, namely, *proper specialization*, *subordination*, *power type of* and *categorization*. Therefore, the definitions of these relations in the formalization that accounts for dynamic classification are most similar to the ones presented in sections 4 and 5, with minor adjustments concerning the quantification of possible worlds (see Definitions D1’, D2’, D3’, D4’, D5’, D6’, D7’and D8’).

$$\forall \mathrm{t1}, \mathrm{t2}\ (\mathrm{specializes}(\mathrm{t1}, \mathrm{t2}) \leftrightarrow (\exists \mathrm{y}, \mathrm{w1}\ (\mathrm{iof}(\mathrm{y}, \mathrm{t1}, \mathrm{w1})) \wedge \forall \mathrm{e}, \mathrm{w2}\ (\mathrm{iof}(\mathrm{e}, \mathrm{t1}, \mathrm{w2}) \rightarrow \mathrm{iof}(\mathrm{e}, \mathrm{t2}, \mathrm{w2})))) \quad \text{(D1’)}$$

$$\forall\ \mathrm{t1}, \mathrm{t2}\ (\mathrm{properSpecializes}(\mathrm{t1}, \mathrm{t2}) \leftrightarrow \big(\mathrm{specializes}(\mathrm{t1}, \mathrm{t2}) \wedge \neg(\mathrm{t1} = \mathrm{t2})\big)) \quad \text{(D2’)}$$

$$\begin{gathered}\forall \mathrm{t1}, \mathrm{t2}\ (\mathrm{isSubordinateTo}\ (\mathrm{t1}, \mathrm{t2}) \leftrightarrow \\ (\exists \mathrm{x}, \mathrm{w1}\ (\mathrm{iof}(\mathrm{x}, \mathrm{t1}, \mathrm{w1})) \wedge \forall \mathrm{t3}, \mathrm{w2}\ (\mathrm{iof}(\mathrm{t3}, \mathrm{t1}, \mathrm{w2}) \rightarrow \exists \mathrm{t4}\ (\mathrm{iof}(\mathrm{t4}, \mathrm{t2}, \mathrm{w2}) \wedge \mathrm{properSpecializes}(\mathrm{t3}, \mathrm{t4}))))) \quad \text{(D3’)}\end{gathered}$$

$$\begin{gathered}\forall \mathrm{t1}, \mathrm{t2}\ (\mathrm{isPowertypeOf}(\mathrm{t1}, \mathrm{t2}) \leftrightarrow \\ (\exists \mathrm{x}, \mathrm{w1}\ \big(\mathrm{iof}(\mathrm{x}, \mathrm{t1}, \mathrm{w1})\big) \wedge \forall \mathrm{t3}, \mathrm{w2}\ \big(\forall \mathrm{t3}\ \mathrm{iof}(\mathrm{t3}, \mathrm{t1}, \mathrm{w2}) \leftrightarrow \mathrm{specializes}(\mathrm{t3}, \mathrm{t2})\big))) \quad \text{(D4’)}\end{gathered}$$

$$\forall t1, t2\ (\text{categorizes}\ (t1, t2) \leftrightarrow (\exists x, w1\ (\text{iof}(x, t1, w1))\forall t3, w2\ (\text{iof}(t3, t1, w2) \rightarrow \text{properSpecializes}(t3, t2)))) \qquad (\text{D5'})$$

$$\forall t1, t2\ (\text{completelyCategorizes}(t1, t2) \leftrightarrow (\text{categorizes}(t1, t2) \wedge \forall w, e\ (\text{iof}(e, t2, w) \rightarrow \exists t3\ (\text{iof}(e, t3, w) \wedge \text{iof}(t3, t1, w))))) \qquad (\text{D6'})$$

$$\forall t1, t2\ (\text{disjointlyCategorizes}\ (t1, t2) \leftrightarrow\ (\text{categorizes}(t1, t2) \wedge \forall w, e, t3, t4\ ((\text{iof}(t3, t1, w) \wedge \text{iof}(t4, t1, w) \wedge \text{iof}(e, t3, w) \wedge \text{iof}(e, t4, w)) \rightarrow t3 = t4))) \qquad (\text{D7'})$$

$$\forall t1, t2\ (\text{partitions}(t1, t2) \leftrightarrow (\text{completelyCategorizes}(t1, t2) \wedge \text{disjointlyCategorizes}(t1, t2))) \qquad (\text{D8'})$$

Further, all the theorems presented in the previous formalization are also valid when considering dynamic classification. The theorems T1-T4 and T7-T9 presented in sections 2 and 3 must be properly adapted considering the use of the world-indexed instantiation relation while the theorems T10 – T15 presented in section 4 can be included in this formalization without modification. The full formalization of MLT supporting dynamic classification can be found in https://github.com/jpalmeida/mlt-ontology.

6.3 A Note on the Identity Conditions of Types

The notion of equality of types is central to account for both Odell's and Cardelli's notions of power type. For example, according to Odell's notion, there is no instance of the "power type" that is equal to the base type. Further, considering Cardelli's sense, (in-)equality is key to establish the uniqueness of the power type for a given base type, as well as the uniqueness of a base type for a given power type. So, which notion of identity condition is adequate in the theory becomes an important issue.

As discussed in [48], there is a spectrum of options for the identity conditions of types, with respect to how finely they are individuated. In an *"infra-coarse"* account, types with the same extension are considered identical. This is what we would call an *"entirely extensional" approach*.

In contrast, in a *"medium-coarse"* account, types "are identical just in case they necessarily have the same extension" [48]. "This seems to transpose the identity conditions for sets into an appropriately intensional key, and this is precisely how identity conditions for properties work in accounts that treat them as intensions" [48]. In such accounts, intensions of types are functions from possible worlds to sets of objects therein [32]. This is one of the approaches that Bealer [8] uses for dealing with "intensional entities". This is what we would call an *"intensional" approach*.

Finally, in a *"ultra-fine"* approach (also referred to as "hyperintensional" approach [48]) types "are individuated almost as finely as the linguistic expressions that express them" [48]. According to “hyperintensional” approaches, two types can be considered distinct even in cases they necessarily have the same extension. For example, if we consider two types *t* and *t’* such that the intension of t’ is formed by a conjunction of the intension of *t* with a trivially “true” statement, adopting an *“ultra-fine”* approach *t* and *t’* must be considered two distinct types.

As discussed by Bealer [8], the medium-coarse and the ultra-fine accounts have each their own value for the intensional conception of types, with different applications. He defends that the medium-coarse approach is ideally suited for treating the modalities (necessity, possibility, impossibility, contingency, etc.), and that the ultra-fine approach is valuable for dealing with intentional matters (belief, desire, perception, decision, etc.). He discusses that the ultra-fine approach, while ideally suitable for the treatment of intentional matters, “has only complicated the treatment of the modalities” [8]. Given the scope of the present paper, we opt for the medium-coarse approach. It allows us to state the impossibility of individuals to have instances (A1) and later (in section 6.2) to deal with types that apply contingently (or non-necessarily) to their instances (e.g., Student, Living Person).

In the formalization presented in section 2, while modality is not formally treated, our choice for the medium-coarse account is reflected in how we stipulate the domain of quantification, which includes all possible types and all their possible instances. The fact that we quantify over all possible entities guides the interpretation of the axioms and definitions, in which quantifiers end up having some modal importance (even if informally). For example, Axiom A2 defines that two types are equal *iff all their possible instances coincide*. In other words, A2 states two types t1 and t2 are equal *iff it is inadmissible for an entity to be an instance of t1 and not an instance of t2* (there is no possible entity that is an instance of t1 and not an instance of t2). Our choice of language here (first-order logics instead of some sort of quantified modal logics) aims at making the theory more accessible.

In section 6.2, we reify possible worlds, and thus, modality is addressed more explicitly in the formalization. Again, here we have opted not to use some sort of modal logic to retain the accessibility of the theory. The axiom that defines equality (A2’) states that two types are equal *iff they have the same instances in all possible worlds*, clarifying that we take the approach that two types are the same *iff they are necessarily coextensional*. Since the extension of a type is world-dependent, it makes sense to talk about the distinction between extension (in a particular world) and intension (across worlds) [32].

7 Related Work

7.1 Power type-based approaches

Two early attempts to address multi-level modeling, namely power types [9, 39] and materialization [44], raised from the identification of patterns to represent the relationship between a class of categories and a class of more concrete entities. The notion of power types was adopted in the object-oriented model community (largely influenced by [39]) and materialization has been developed in the database community. Despite the different origins, power type and materialization are based on similar conceptualizations [5] and addressing the same concerns [17]. Both approaches establish a relationship between two types such that the instances of one are specializations (subtypes) of another.

Odell [39] defined the concept of power type informally using regular associations between a class representing the power type and a base class. This differs from our approach because cross-layer relations between types (*is power type of, categorizes* and *partitions*) have specialized semantics. This allows us to prescribe rules for the domain models that use these relations following the axioms in the theory.

Similarly to Odell [39], Gonzalez-Perez and Henderson-Sellers [17] use an association labeled "partitions" between a power type and a base type (called a "partitioned type" in their terminology). The authors illustrate their technique with a diagram in which "partitions" is modeled as a one-to-many association between "Task" and "TaskKind", meaning that every instance of the partitioned type ("Task") is linked to exactly one instance of the powertype ("TaskKind"). In the sequel, they discuss that the "*partitions* association possesses instantiation semantics", and that, because of this, "Task" is a special instance of "TaskKind" (the most generic kind of task). However, if "Task" itself is an instance of "TaskKind", then the "partitions" association cannot be a one-to-many association between "Task" and "TaskKind". This is because all instances of subtypes of "Task" are also instances of "Task", and thus instances of at least *two* "TaskKinds" (one of which is "Task" itself). The source of the difficulty seems to lie in that their "partitions" association is semantically overloaded, conflating two underlying notions: (i) the fact that "TaskKind" *partitions* "Task", and (ii) the implied consequence that instances of "Task" are instances of instances of "TaskKind" (which in our theory is reflected in the *instance of* relation between "Task" as specialization of "Individual" and "TaskKind" as a specialization of "First-Order Type"). The modeler is free to determine whether "Task" itself is an instance of "TaskKind" (in which case he/she would replace (i) with the fact that "TaskKind" *is a powertype of* "Task"). Note that the elements of our theory help us to identify the semantic overload, provide an explanation for the conceptual issue in this power type based approach, and offer alternatives to express the modeler's intended conceptualization.

The UML 2.4.1 specification [41] attempts to cover the needs of multi-level modeling by including a *powertype association* that relates a classifier (power type) to a generalization set composed by the generalizations that occur between the base classifier and the instances of the powertype. Because of its dependence on the generalization set construct, the 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 would rule out simple models that are possible in our approach, e.g., one defining “employee type” as a powertype of “employee”, without forcing the modeler to define specific instances for “employee type”. While our theory necessitates the existence of entities for any type, and hence necessitates the existence of instances for “employee type”, it does not require these instances to be modeled explicitly, which is the case of the UML because of its choice to base the power type pattern in a structure that uses generalization sets.

Further, while the *complete categorization* relation is similar to the *isCovering* attribute of *GeneralizationSets* of the UML metamodel, there is an important distinction. The attribute *isCovering* refers to whether all instances of the general classifier are instances of at least one of the specific classifiers that are explicitly modeled in the *GeneralizationSet*. In contrast, *completelyCategorizes* is a semantic notion that is independent of what is represented explicitly in a model; when a higher-order type is related to a base type through this relation, all instances of the base type will be instances of at least one of the types that properly specialize the base type.

A semantic mapping of UML’s *isCovering* attribute in terms of our theory is simple when *isCovering* is true, in which case the higher-order type *p completely categorizes* the base type *t*. However, when *isCovering* is false, the semantic mapping is more involved, and the model can have two alternative interpretations: (i) *p categorizes t* and there are instances of *p* not represented in the model (it is not possible to determine whether *p completely categorizes t*); or (ii) *p categorizes t* and all instances of *p* are represented, but some instances of *t* do not instantiate any of the instances of *p* (thus, we conclude that *p* does not *completely categorize t).* The lack of expressiveness of UML to distinguish these interpretations seems to stem from the fact that UML conflates what we mean to capture with the *categorization* relation with whether the *model enumerates all instances of a higher-order type* (i.e., the “power type” in UML’s terminology).[7]

The notion of *power type* introduced by Odell [39] in the object oriented community differs from the concept coined earlier by Cardelli [9] since the latter is derived directly from the mathematical notion of *power set* while the former may be used more loosely as we discussed in section 4. The theory presented here is able to account for both definitions formally, revealing their differences. It covers the expressiveness of both approaches through formally-defined structural cross-level relations (*power type of, categorization* and *partitioning*). Further, it allows us to show that a higher-order type that is related to a base type through the *categorization* relation is necessarily a specialization of the power type of that base type. Thus, the power type of a base type is the most abstract higher-order type related to a base type.

[7] For the sake of simplicity we have assumed here that the classes are not abstract. The semantic mapping becomes even more involved in the presence of abstract classes.

7.2 Clabject and Deep Instantiation based approaches

The concept of power type is founded on the notion that “instances of types can also be types” [39]. Motivated by a similar observation, Atkinson and Kühne [2] coined the term *clabject*, emphasizing that every instantiable entity has both a type (or class) facet and an instance (or object) facet which are equally valid [2]. This notion is valuable to our theory. The basic types of MLT, except the higher order one, may be considered clabjects. For example, “Individual” is instance of “First-Order Type” (its instance facet) and a type for all entities that are not types (its type or class facet). (Note that, individuals in MLT (instances of “Individual”) have no class facet, and thus should not be referred to as clabjects.)

In [2], Atkinson and Kühne argue that a multi-level modeling framework should adhere to two fundamental principles: support for the *clabject* notion and *strict metamodeling*. *Strict metamodeling* [7] assumes that each element of a level must be an instance of an element of the level above. Although our theory is not focused on metamodeling, we follow this principle with respect to the instantiation relation, and every entity in our domain of enquiry is instance of exactly one of the basic types and every entity can only be instance of entities at one order higher (all entities that have no instances are instances of “Individual”; “Individual” and all its specializations are instances of “First-Order Type”, and so on.) They also discuss that “some kind of ‘trick’ is needed at the top level”. The ‘trick’ we used in our theory is that the highest order basic type is not instance of anything, since entities with higher order are not considered (see axiom A6). Alternatively, an infinite number of basic types may be considered at ever increasing orders, in which case a ‘trick’ at the top of the classification scheme would not be required. We have opted instead for a finite number of basic types to avoid necessitating the existence of an infinite number of levels, which would be an unnecessary ontological commitment for all conceptual modeling applications we have considered so far.

Atkinson and Kühne propose in [3] the Orthogonal Classification Architecture (OCA) to address the need of considering two different kinds of instantiation: the linguistic instantiation and the ontological instantiation. Whereas linguistic instantiation is used to define the relations between domain entities and linguistic constructs, ontological instantiations relate domain entities to other domain entities. For example, considering the UML class diagram having a *class* called *Collie* and an *object* called *Lassie,* we can identify two linguistic instantiations, namely, *Collie* is an (linguistic) instance of *class* and *Lassie* is an (linguistic) instance of *object.* We may also consider an ontological instantiation since *Lassie* is an (ontological) instance of *Collie* [3]. The distinction is very important to determine the scope of our theory: we are concerned solely with ontological instantiation as it is applied across multiple levels. For example, the instantiation relation that holds between *individual* and *first-order type* as well as the one that holds between *computer* and *first-order type* are both ontological since both are concerned with the nature of the involved concepts and none of them is related to linguistic issues. Aspects referring to the relation between ontological and linguistic issues are out of the scope of this work.

Atkinson and Kühne have also proposed the notion of *deep instantiation* [5, 6] as a means to provide for multiple levels of classification whereby an element at some level can describe features of elements at each level beneath that level. It is based on the idea of assigning to *clabjects* and *fields* (attributes and slots) a *potency* which defines how deep the instantiation chain produced by that *clabject* or *field* may become. When a *clabject* is instantiated from another *clabject* the potencies of the created *clabject* and of its *fields* are given by the original *clabject* and *fields* potencies decremented by one. Objects have potency equal to zero indicating they cannot be instantiated. If the potency of a *field* becomes zero then a value can be assigned to that *field.* For example, we could define a *clabject mobile phone model* with an attribute *IMEI* assigning a potency of 2 to both the type and the attribute. Therefore, instances of *mobile phone model* would be *clabjects* in which *IMEI* attribute would have potency of 1. Instances of instances of *mobile phone model* have a value assigned to *IMEI*, since its potency would reach zero.

The authors consider that the main benefit of a *deep instantiation based* approach is to reduce "accidental complexity" in domain models since it supports multi-level modeling without the need of introducing types to the models only "because of the idiosyncrasies of a particular solution to deep characterization" [5]. They argue that *power type* based solutions force the modelers to add unneeded types to the model. For instance, considering the cited example of *mobile phone model*, using *power types* the modeler would be forced to represent the concept of *mobile phone*. Using deep instantiation, the modeler could define the *mobile phone* properties (e.g. *IMEI)* as properties of *mobile phone type* having potency of 2, being free not to represent the concept of *mobile phone*.

While the deep instantiation approach can reduce the number of entities represented in a model, this strategy should be used with parsimony. Important consequences of omitting base types in the current deep instantiation approach are that the modeler becomes unable to express whether the instances of a higher-order type (mobile phone model in previous example) are disjoint and/or covering types and we are also prevented from determining metaproperties (such as e.g., rigidity) of the base type (mobile phone in this case). Further, as discussed in [21], conceptual models should always include kinds that define the principle of identity of individuals (in the example this type is mobile phone). If these types are omitted (and incorporated into higher-order types by using the notion of potency), the source of the principle of identity becomes hidden.

It is worth noticing that the deep instantiation approach allows the modeler to represent the base type if it is deemed desirable. However, if the modeler decides to represent the base type, the approach does not provide constructs to represent the relation between it and the higher-order type, not distinguishing thus between the different possible kinds of cross-level relations. As a consequence, the approach "as is" does not provide mechanisms to check if the rules concerning these relations are respected, e.g., to guarantee that all instances of the higher-order type ("Mobile Phone Model") specialize the base type ("Mobile Phone"). We believe that the relations and rules we discuss here could be used to further evolve the deep instantiation approach.

In addition to having mechanisms aimed at simplifying the models by omitting base types, some recent deep instantiation approaches [25] also support the representation of a particular kind of regularity attribute by using a combination of the notions of attribute durability and mutability. The durability of an attribute indicates how far the attribute spans in an instantiation tree. The mutability of an attribute defines how often the attribute value can be changed over the instantiation tree. Consider for example a class such as "MobilePhoneModel" with potency 2. An attribute "screenSize" with durability 2 and mutability 1 will be given a value at the first instantiation (e.g., stating that the "Iphone5" has "screenSize" equals to 4 inches), and that value will determine the value of "screenSize" for the instances of instances of "MobilePhoneModel" (thus, all Iphone5s have a screen size of 4 inches). In our view, this representation captures the constraint relating an attribute of a first-order type ("screenSize") and a regularity attribute of a second-order type ("instancesScreenSize") as a single attribute with durability 2 and mutability 1 in a clabject with potency 2. This is a useful language mechanism for this particular kind of regularity attribute, which fully determines the value of the lower-level attribute. Unfortunately, this representation strategy is only capable of capturing the constraints involving regularity attributes that determine the exact value that must be assigned to attributes in the lower order. It is not applicable, for example, to capture the constraint involving the regularity attributes "instancesMinStorageCapacity" and "instancesMaxStorageCapacity" of "MobilePhoneModel": an instance i of "MobilePhone" must have assigned to the attribute "storageCapacity" a value equal or higher than the value assigned to "instancesMinStorageCapacity" and equal or lower than the value assigned to "instancesMaxStorageCapacity" of the "MobilePhoneType" instantiated by i. Thus, since the values of "instancesMinStorageCapacity" and "instancesMaxStorageCapacity" are not directly reflected in the value of a mobile phone attribute, they would be given durability and mutability of 1.

In an analysis of deep instantiation, Neumayr et al., [37] observe that the approach is unable to capture certain domain scenarios in which a clabject is related to other clabjects at different instantiation levels. For example, consider a scenario in which every instance of "MobilePhoneModel" has a "designer" being an instance of "Person" and every instance of an instance of "MobilePhoneModel" (i.e., every instance of "MobilePhone") has an "owner" which is also an instance of "Person". In this scenario the type "Person" should have a relation with "MobilePhoneModel" (called "designer") and another relation with "MobilePhone" (called "owner"). Considering that "Person" and "MobilePhone" are in the same level, the "owner" relation does not cross level boundaries. Nevertheless, "MobilePhoneModel" is placed in one level higher, and thus, the "designer" relation is crossing level boundaries, which is not allowed in that approach. Because of that, the authors introduce a Dual Deep Instantiation (DDI) approach distinguishing between source potency and target potency. An association thus becomes characterized by two potency numbers. Thus, in the aforementioned example, the “designer” relationship between “Person” and “MobilePhoneModel” would have both source and target potencies of 1 whereas the “owner” relationship would be defined having source potency of 1 and target potency of 2 (meaning that ownership relations hold between instances of “Person” and instance of instances of “MobilePhoneModel”). Another approach that allows for the representation of this kind of domain scenario is discussed in [4]. The approach is based on the definition of the so-called “metamodeling spaces”, each of which defines a separate set of instantiation levels. Levels in one metamodeling space are independent of levels in other spaces. As a consequence, an element in a particular metamodeling space *S* may be simultaneously related to elements in different levels as long as the target elements are not in *S*. For example, “Person” in a space *P* could be related to both “Mobile Phone” and “Mobile Phone Model” placed in different levels in another space *M*. Differently from [37] and [4], our approach accommodates the domain scenario without a special mechanism, since relations between elements at different orders are allowed. In the example considered, we would represent the omitted base type “MobilePhone” defining that “MobilePhoneModel” *partitions* “MobilePhone”, capturing thus, the fact that every instance of “MobilePhone” is instance of one instance of “MobilePhoneModel”. The “designer” relationship would be placed between “MobilePhoneModel” and “Person” whereas the “owner” relationship would be defined between “MobilePhone” and “Person”.

In [36], another multi-level modeling approach that applies the notion of deep instantiation is proposed. The focus of this approach is also on reducing “unnecessary complexity”, improve readability and simplify maintenance and extension. The approach is founded in the concepts of m-objects and m-relationships. M-objects encapsulate different levels of abstraction that relate to a single domain concept. Analogously, m-relationships describe “relationships between m-objects at multiple level of abstraction”. An m-object can concretize another m-object. The concretize relationship comprises classification, generalization and aggregation relationships between the levels of an m-object [36]. We observe that this is a semantic overload between three relationships of quite different ontological nature, which could affect the understandability and usability of the approach.

Telos [34] is a knowledge representation language that supports the representation of types having other types as instances (i.e. clabjects). Roughly 30 axioms are defined to formalize Telos’ principles for instantiation, specialization, object naming and attribute definition [30]. Telos supports multi-level modeling through its notion of type, and, similarly to MLT it accepts relations between types in different levels. In contrast with MLT, it does not elaborate on the nature of cross-level relations between higher-order types and base types. Further, it does not employ systematically the powertype pattern, although we consider it would be possible to extend the Telos built-in support by using its features of user-defined constraints and rules to formally define the cross-level structural relations proposed in MLT.

Besides the so far mentioned initiatives, many other works focusing on deep instantiation based approaches can be found in the literature, proposing alternative formalizations for it (e.g. [46]), exploring its uses in different contexts (e.g. [28, 29]), and proposing tools for automated support (e.g. [1, 27]). These works focus on deep instantiation, which illustrates its wide acceptance as a basic mechanism for multi-level modeling approaches. However, none of these approaches aim at providing a semantic account that can be used to explain regularity attributes in deep instantiation and that supports the power type pattern and its variations.

8 Final Considerations

In this paper we have presented a well-founded theory for conceptual multi-level modeling. The theory is formally defined using first-order logic and its consistency is verified using a lightweight formal method. Both the basic types and the structural relations defined in the theory are founded on the basic notion of (ontological) instantiation, which is applied regularly across levels, organizing the entities of the domain of enquiry in strictly stratified orders. We have shown how the elements of the theory can be used as foundations for a domain theory: domain types instantiate and specialize the basic types of the theory.

To verify the consistency of our theory we have used Alloy [24]. The axioms of our theory were represented as facts and the theorems were defined as assertions in an Alloy module. It allowed us to verify the satisfiability of our theory, to conduct some model simulations and to verify the theorems whose informal proofs have been discussed in the paper.[8]

Using the structural cross-level relations defined in the theory (*powertype of*, *categorization*, *partitioning*), we are able to account for the different notions of power type in the literature, as well as to contrast and relate them. Since these relations are ultimately explained in terms of instantiation between entities of adjacent levels, the consequence of our account of power types is that we formally harmonize power type and clabject-based approaches.

With respect to intra-level relations, we define the "ordinary" specialization relation and a *subordination* relation between higher-order types of the same order. Subordination allows for the creation of expressive multi-level models; subordination between higher-order types implies specialization between instances of the types related by subordination. An example of the usefulness of the subordination relation is shown in the biological taxonomy domain, in which taxonomic ranks (instances of "Second-Order Type") are related by subordination in a sequence (with lower ranks subordinated to higher ranks). This ensures the taxonomy at the first-order level has an adequate structure (a taxonomic tree).

[8] The full specification of the theory in Alloy can be found in https://github.com/jpalmeida/mlt-ontology

In order to facilitate the readers' first contact with MLT, we have opted to supress two direct extensions of the theory in the early part of this paper: (i) the support for non-rigid types, and (ii) the generalization of the notion of order to support an infinite number of classification levels. With respect to (i) this extension is presented in section 6.2. It allows instantiation to be contingent, thereby enabling dynamic classification, which is an important feature for conceptual modeling [21]. With respect to (ii), axioms A3, A4 and A5 would give way to an inductive definition for a basic type T_{i+1} based on the definition of the basic type at an immediately lower order T_i. The "disjointness" axiom (A6) would be modified accordingly.

The whole theory presented here is built up from an 'opaque' notion of instantiation, i.e., using instantiation as a primitive notion and not appealing to the 'internals' of intensions. The resulting theory is thus independent of any modeling choices or ontological commitments concerning the nature of 'intensions' of types. Naturally, this could be worked out in an extension of this work. To formally discuss the nature of 'intensions' of types, one could either opt for using a higher-order logics (which we avoid here intentionally since we aim at a more approachable formalization) or to reify and treat the intensions of types as structured elements with 'parts' or "constituents pretty much like the linguistic expressions that we use to speak about them" [48]. The adequacy of these approaches is an issue for further investigation.

It is important to stress that it is not our intention in this paper to propose a multi-level conceptual modeling language. Instead, we focus on the concepts that would constitute an adequate semantic domain for such a language. The theory we propose can be considered a reference top-level ontology for types, with the main purpose of clarifying key concepts and relations for multi-level conceptualizations.

As discussed in [21], a reference ontology can be used to inform the revision and redesign of a modeling language, first through the identification of semantic overload, construct deficit, construct excess and construct redundancy, but also through the definition of modeling patterns and semantically-motivated syntactic constraints [11]. This has been fruitful in the past in the revision of the UML, resulting in the OntoUML profile for conceptual modeling [21]. Thus, a natural application for this work is to inform the (re-)design of a well-founded multi-level conceptual modeling language. Some earlier results to that extent are presented in section 4, showing: (i) how theorems of the theory reveal useful syntactic constraints for multi-level domain models; and (ii) how patterns of domain entities that are admissible by the theory can be reflected in modeling patterns. We have also been able to spot a deficiency in the UML given its reliance on the construct of generalization set to represent the power type pattern. Further, we have been able to identify cases of semantic overload in the power-type based technique presented in [17], and in the m-objects approach [36]. Recently, Recker et al. [45] reported results from a study with 528 modelers demonstrating that "users of conceptual modeling grammars perceive ontological deficiencies to exist and that these deficiency perceptions are negatively associated with usefulness and ease of use of these grammars". This highlights the potential practical implications of our theory.

We are currently working on an extension of the Unified Foundation Ontology (UFO) [21] to fully incorporate the theory presented in this paper. The current version of UFO only counts with an informal notion of higher-order universal, with no associated formalization. The theory would serve as the top-most layer of UFO, and the typology of universals of UFO would be incorporated as specializations of "First-Order Type", including RigidUniversal, Anti-RigidUniversal, Category, Kind, Role, Phase, etc. Further, "Individual" would be specialized into Endurant, Moment, Event, Action, etc., leveraging important conceptual distinctions of UFO. The revision of UFO to incorporate this theory will give us a sound basis to improve the formalization of ontologies based on UFO (e.g., the core ontology for services called UFO-S [35] and the organizational ontology called O3 [43]) since their conceptualizations span multiple levels of classification.

Acknowledgments

This research is partly funded by the Brazilian Research Funding Agencies CAPES, CNPq (grant numbers 311313/2014-0, 485368/2013-7, 461777/2014-2) and FAPES (grant number 69382549).

References

1. Atkinson, C.; Gerbig, R.: Melanie: multi-level modeling and ontology engineering environment. In: Proc. of the 2nd International Master Class on Model-Driven Engineering Modeling Wizards - MW ’12. New York, USA (2012)
2. Atkinson, C., Kühne, T.: Meta-level Independent Modeling. International Workshop “Model Engineering” (in conjunction with ECOOP’2000), Cannes, France (2000)
3. Atkinson, C., Kühne, T.: Model-driven development: a metamodeling foundation. IEEE Software. 20(5), 36–41 (2003)
4. Atkinson, C., Kühne, T.: Processes and Products in a Multi-level Metamodeling Architecture. Int. Journal of Software Engineering and Knowledge Engineering, 11(6), pp 761-784 (2001)
5. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software & Systems. Modelling, 7(3), pp 345-359. Springer-Verlag (2008)
6. Atkinson, C., Kühne, T.: The Essence of Multilevel Modeling. In: Proc. Of the 4th International Conference on the Unified Modeling Language, pp. 19-33. Toronto, Canada (2001)
7. Atkinson, C.: Metamodelling for Distributed Object Environments. First International Enterprise Distributed Object Computing Workshop (EDOC’97). Brisbane, Australia (1997)
8. Bealer, G.: Quality and Concept. Clarendon Press, Oxford (1982)
9. Cardelli, L.: Structural Subtyping and the Notion of Power Type. In Proc. Of the 15th ACM Symposium of Principles of Programming Languages, pp. 70-79 (1988)
10. Carvalho, V. A., Almeida, J.P.A.: A Semantic Foundation for Organizational Structures: A Multi-Level Approach. In: Proc. of the Enterprise Computing Conference (EDOC2015). (2015)
11. Carvalho, V. A., Almeida, J.P.A., Guizzardi, G.: Using Reference Domain Ontologies to Define the Real-World Semantics of Domain-Specific Languages. In: Proc. 26th International CAiSE Conference (CAiSE 2014), Heidelberg: Springer, 2014. pp. 488-502 (2014)
12. 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: 35th Intl. Conf. on Conceptual Modeling (ER 2015), pp. 119-133 (2015)
13. Chen, P. P.: The Entity-relationship Model: Toward a Unified View. ACM Transactions on Database Systems, 1 (1), pp. 9-36. (1976)
14. Coquand, T.: Type Theory, The Stanford Encyclopedia of Philosophy (Fall 2014 Edition), Edward N. Zalta (ed.), URL = http://plato.stanford.edu/archives/fall2014/entries/type-theory/ (2014)
15. Ereshefsky, M., Species, The Stanford Encyclopedia of Philosophy (Spring 2010 Edition), Edward N. Zalta (ed.), URL = http://plato.stanford.edu/archives/spr2010/entries/species/ (2010)
16. Eriksson. O., Henderson-Sellers, B., Ågerfalk, P. J.: Ontological and linguistic metamodeling revisited: A language use approach. Information and Software Technology, 55(12), pp. 2099-2124. Elsevier (2013)
17. Gonzalez-Perez, C., Henderson-Sellers, B.: A powertype-based metamodelling framework. Software & Systems. Modelling, 5(1), 72-90. Springer-Verlag (2006)
18. Guarino, N., Welty, C.: Evaluating Ontological Decisions with OntoClean. In Communications of the ACM, 45(2), pp.61-65. (2002)
19. Guarino, N.: The Ontological Level. In: R. Casati, B. Smith and G. White (eds.), Philosophy and the Cognitive Science, pp. 443-456. Holder-Pivhler-Tempsky, Vienna (1994)
20. Guarino, N., Guizzardi, G.: “We Need to Discuss the Relationship”: Revisiting Relationships as Modeling Constructs. In: Proc. 27th International CAiSE Conference (CAiSE 2015). pp. 488-502 (2015).
21. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. University of Twente, Enschede, The Netherlands (2005)
22. Guizzardi, G. et al.: Towards an Ontological Analysis of Powertypes. In: Proc. of the International Workshop on Formal Ontologies for Artificial Intelligence (FOFAI 2015), 24th International Joint Conference on Artificial Intelligence. (2015)
23. Henderson-Sellers, B.: On the Mathematics of Modeling, Metamodelling, Ontologies and Modelling Languages. Springer (2012)
24. Jackson, D.: Software Abstractions: Logic, Language and Analysis. The MIT Press, (2006)
25. Kennel, B.: A Unified Framework for Multi-Level Modeling. University of Mannheim (2012)
26. Kühne, T.: Contrasting Classification with Generalisation. In: Proc. of the 6th Asia-Pacific Conference on Conceptual Modeling. Wellington, New Zealand (2009)
27. Lara, J. de, Guerra, E.: Deep Meta-modelling with MetaDepth. In: Proc. of the 48th International Conference, TOOLS 2010. Málaga, Spain (2010)
28. Lara, J. de, Guerra, E., Cuadrado, J. S.: When and How to Use Multilevel Modelling. ACM Transactions on Software Engineering and Methodology, v. 24, n. 2, p. 1–46, 23 (2014)

29. Lara, J. et al.: Extending Deep Meta-Modelling for Practical Model-Driven Engineering. The Computer Journal, (2013)
30. Jarke, M., Gallersdörfer, R., Jeusfeld, M. A., Staudt, M.: ConceptBase - A Deductive Object Base for Meta Data Management. J. Intell. Inf. Syst. v 4(2), pp. 167-192 (1995)
31. Mayr, E., The Growth of Biological Thought: Diversity, Evolution, and Inheritance Belknap Press, (1982).
32. Montague, R.: Formal Philosophy: Selected Papers of Richard Montague. In: Thomasson, R. (Eds.). Yale University Press (1974).
33. Mylopoulos, J.: Conceptual Modeling and Telos. In: Loucopoulos, P., Zicari, R. (Eds.), Conceptual modeling, databases and CASE, pp. 49-68, Wiley(1992)
34. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: Representing Knowledge About Information Systems. ACM Trans. Inf. Syst, v. 8(4), pp. 325-362 (1990)
35. Nardi, J.C., Falbo, R., Almeida, J.P.A., Guizzardi, G., Ferreira Pires, L., van Sinderen, M., Guarino, N.: Towards a Commitment-Based Reference Ontology for Services. In: Proc. 17th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2013), IEEE Computer Society Press, pp. 175-10 (2013).
36. Neumayr, B., Grün, K., Schrefl, M.: Multi-level domain modeling with m-objects and m-relationships. In: Proc. of the 6th Asia-Pacific Conference on Conceptual Modeling. Wellington, New Zealand (2009)
37. Neumayr, B., Jeusfeld, M. A., Schrefl, M., Schütz, C.: Dual Deep instantiation and It ConceptBase Implementation. In: Proc. 26th International CAiSE Conference (CAiSE 2014), Heidelberg: Springer, 2014. pp. 503-517 (2014)
38. Neumayr, B., Schrefl, M., Thalhiem, B.: Modeling Techniques for Multi-Level Abstraction. In: Kaschek, R., Delcambre, L. (eds.). LNCS, vol. 6520, pp 68-92. Springer, Heidelberg(2011)
39. Odell, J.: Power types. In: Journal of Object-Oriented Programing, 7(2), pp. 8-12.(1994)
40. Olivé, A.: Conceptual Modeling of Information Systems. Springer (2007)
41. OMG : UML Superstructure Specification – Version 2.4.1 (2011)
42. OMG: Meta Object Facility (MOF) Core Specification - Version 2.4.1 (2013)
43. Pereira, D., Almeida, J.P.A.: Representing Organizational Structures in an Enterprise Architecture Language. In: Proceedings of the 6th Workshop on Formal Ontologies meet Industry (FOMI 2014), Rio de Janeiro (2014).
44. Pirotte, A., Zimanyi, E., Massart, D., Yakusheva, T.: Materialization: a powerful and ubiquitous abstraction pattern. In: Bocca, J.,Jarke, M., Zaniolo, C. (eds.) Procs. 20th Int. Conf. Very Large DataBases (VLDB ’94) pp. 630–641 (1994)
45. Recker, J., Rosemann, M., Green, P., Indulska, M.: Do Ontological Deficiencies in Modeling Grammars Matter?. In MIS Quarterly, 35 (1): pp. 1–9. (2011)
46. Rossini, A. et al.: A formalisation of deep metamodelling. Formal Aspects of Computing, v. 26, n. 6, pp. 1115–1152 (2014)
47. Steinberg, D., Budinsky, F.: EMF: Eclipse Modeling Framework. 2nd Edition, Addison-Wesley Professional (2008).
48. Swoyer, C., Orilia, F.: Properties. In: Zalta, E. N. (eds.). The Stanford Encyclopedia of Philosophy (Fall 2014 Edition), URL = http://plato.stanford.edu/archives/fall2014/entries/properties/ (2014)
49. W3C.: OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (Second Edition), URL = https://www.w3.org/TR/2012/REC-owl2-syntax-20121211 (2012)