This is a post-print. The original paper appeared in Proc. 35th Intl. Conf. on Conceptual Modeling (ER 2016), Gifu, Japan, LNCS 9974, Springer-Verlag, 2016, pp. 198-211, doi 10.1007/978-3-319-46397-1_15.

DeepTelos: Multi-level Modeling with Most General Instances

Manfred A. Jeusfeld and Bernd Neumayr

University of Skövde, Sweden
manfred.jeusfeld@his.se

Johannes Kepler University Linz, Austria
bernd.neumayr@jku.at

Abstract. Multi-level modeling aims to reduce redundancy in data models by defining properties at the right abstraction level and inheriting them to more specific levels. We revisit one of the earliest such approaches, Telos, and investigate what needs to be added to its axioms to get a true multi-level modeling language. Unlike previous approaches, we define levels not with numeric potencies but with hierarchies of so-called most general instances.

Keywords: multi-level modeling, Telos, meta modeling

1 Introduction

Multi-level modeling [1,13] (or Deep Modeling) aims at reducing accidental complexity [2] in software, data, and domain models by utilizing abstraction levels to express model statements only once rather than repeating them multiple times, e.g., defining a property `listPrice` for product models or a property `owner` for individual products. Early approaches were materialization [19] and power types [18]. Many current approaches [6,20,12,16] to multi-level modeling assign potencies to relationships, attributes, and classes. Potencies constrain how many times the respective concept can be instantiated until it can no further be instantiated.

Telos [14] is a metamodeling language based on the single concept of *proposition* to represent any model elements, regardless of its abstraction level. ConceptBase [8] is an implementation of the O-Telos [9] variant of Telos that maps all language axioms to Datalog. It shares the simplicity of Telos by solely using a single data structure of propositions to represent objects, classes, attributes, relationships, instantiations, and specializations. The 30+ axioms are forming the rules for propositions, in particular how instantiation, specialization and attribution/relationship interplay.

Telos pioneered core ideas of multi-level modeling. First, Telos models come with an *unrestricted number of meta-levels*. Second, there is no separation between classes and objects: a model element (later referred to as *clabject*) may act both as object and as class, and possibly also as metaclass, and so forth. Third, the reliance on a single language construct, proposition, which is instantiated by model elements at all modeling levels, pioneered what was later referred to as *orthogonal classification architecture*.

Telos, however, lacks a crucial feature of multi-level modeling, namely full support for *deep characterization*. That is, in Telos, there is no easy way to specify with a metaclass a property that is instantiated by the instances of the instances of the metaclass. For example, there is no obvious way to specify that all instances of all instances of `ProductCategory` have a property `listPrice` and their instances in turn have a property `owner`.

The contribution of this paper is to extend Telos with *most general instances* (MGI), a language construct for deep characterization. The result is DeepTelos, a language and system for *Deep Metamodeling* (combining metamodeling and multi-level modeling), akin to MetaDepth [12]. What sets DeepTelos apart are the strenghts inherited from Telos and ConceptBase:

- simplicity and conceptual clarity
- formal semantics expressed and implemented in Datalog
- rich query and query optimization facilities

In the remainder of the paper we introduce, in Section 2, the relevant O-Telos axioms. The analysis of the axioms reveals that O-Telos as such is unable to support deep characterization by its existing axioms. In Section 3 we propose the simple yet powerful construct of *most generic instances* (MGI) to support deep characterization within the axiomatic boundaries of Telos. Telos extended with MGIs allows to combine "lingustic" metamodeling, e.g. modeling the entity-relationship model, and "ontological" multi-level modeling, e.g., specifying product hierarchies with multiple levels. In Section 4 we present details about the implementation of the approach and the implementation of the running example. In Section 4 we discuss the approach and related work.

1.1 Running example

The running example is derived from the example discussed in [15]: There are product categories such as car models and phone models, also called product models. Product models subsume products, which themselves subsume individual products. Product categories have persons as category managers. Car models have attributes like the number of doors. Product models have a list price. Finally, products have a person as owner.

The example highlights that some concepts have both the nature of a class (defining attribute and relation types for their instances) and of an object (instantiating attribute and relation types). Such concepts are commonly referred to as *clabject* [1,7]. In classical two-level models, one would have to separate the class and object flavors into different objects, e.g. using the powertype pattern [4]. Multi-level modeling aims at avoiding this separation by regarding each class also as an object that can have its own properties.

2 Telos

Telos was originally developed for requirements modeling [5] but later mostly applied for the design of interrelated modeling languages [17,10], i.e. for the linguistic flavor

of metamodeling. Its O-Telos axioms [9] are defining the interplay between instantiation, specialization, and properties (subsuming attributes and relationships). A proposition in Telos is a quadruple $P(o, x, n, y)$ where o is its identifier, x its source, n its label, and y its target. The components x and y are identifers of either the same proposition (then it has the form $P(o, o, n, o)$ and is displayed as a node) or of some other propositions (then it is a link between the other propositions). If the label is 'in', then it is an explicit instantiation. If the label is 'isa', then it is an explicit specialization, otherwise it is a relation between two proposition. Telos does not distinguish objects and values. Hence attributes are just relations between propositions, the one being interpreted as an object, the other being intepreted as a value. Propositions are then used to derive the predicates *In* and *Isa*:

$$\forall\ o, x, c\ P(o, x, in, c) \Rightarrow In(x, c) \tag{1}$$

$$\forall\ o, c, d\ P(o, c, isa, d) \Rightarrow Isa(c, d) \tag{2}$$

Instantiation and specialization interplay via a number of axioms. The first one is about inheritance of class membership, see also left half of figure 1:

$$\forall\ x, c, d\ In(x, c) \wedge Isa(c, d) \Rightarrow In(x, d) \tag{3}$$

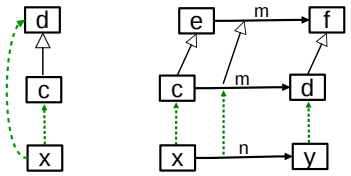

Fig. 1. Instantiation and specialization in Telos

Instantiations are displayed as broken directed links and specializations as directed links with white arrows heads. The instantiation of relations such as the relation labelled m between c and d in the right half of figure 1 requires that both the sources and targets of the instance with label n are synchronously instantiated:

$$\forall\ o, x, n, y, p\ P(o, x, n, y) \wedge In(o, p) \Rightarrow \exists\ c, m, d\ P(p, c, m, d) \wedge In(x, c) \wedge In(y, d) \tag{4}$$

The upper half of figure 1 refers to an analogous constraint on the specialization of relations. This synchronous semantics of instantiation and specialization made Telos appear unsuitable for true multi-level modeling, at least on first sight. To show this consider a Telos metaclass `Product` that has an attribute `serialnumber` whose value

is an integer number. An instance of the attribute `serialnumber` has to instantiate the source of the attribute to an instance of `Product`, i.e., a simple class, and the destination to an instance of integer. The latter one cannot be further instantiated, the former one is a class. Hence, we cannot define serial numbers of actual products in this manner.

On the other hand, Telos does not distinguish classes from objects. Instead it links them via the instantiation predicate $In(x, c)$. Telos is also agnostic of any pre-defined abstraction levels. Abstractions levels rather follow from chains of instantiation facts such as

$$\ldots, In(x, c), In(c, mc), In(mc, mmc), \ldots$$

Such chains are un-restricted on both sides, i.e. the object x could also have instances, and the object mmc could have classes.

3 Most general instances

Our running example mentions the concepts product categories, product models, and products. Each of them has describing properties such as the ownership of a product, or the number of doors of a car model. Intuitively, products, product models, and product categories are at different abstraction levels, but how can this be expressed in Telos that has no builtin abstraction levels? We propose the construct of *most general instances* to formalize the relationship between the concepts. The most general instance of a class c is a class m that has all instances of c as subclasses:

$$\boxed{\forall\ x, c, m\ \ In(x, c) \wedge IN(m, c) \Rightarrow Isa(x, m)} \tag{5}$$

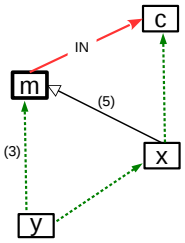

Fig. 2. Most general instance

Figure 2 visualizes the construct. The class c has the (regular) instance x and the most general instance m. The axiom (5) then demands that x is a subclass of m. The axiom is similar to axiom (3). The difference is that it does not derive instantiations but specializations. The predicate $IN(m, c)$ defines m to be the most general instance of c. This should not be confused with the instantiation predicate $In(x, c)$. The most general instance of a class is usually not an instance of the class itself. It rather

is a *proxy* of the class at the abstraction level below the class. It has all instances of all instances of class c as its instances. This is defined via the class membership inheritance axiom (3). In figure 2 the instantiation $In(y, m)$ is derived via axiom (3).

A most general instance is placed on the top of generalization hierarchies. One may argue that such a class must be abstract, i.e. that it should not have any instance that does not occur in one of its proper subclasses. We leave this open in order to minimize the set of additional axioms. The original axiom set [9] plus the axiom (5) is referred to as *DeepTelos*, since it allows to use multi-level modeling with Telos as discussed subsequently.

3.1 Linguistic use of most general instances

Since Telos was originally developed for linguistic metamodeling, we apply the new construct first to the entity-relationship diagramming language (ERD). It features as constructs entity types, relationship types, role links between relationship types and entity types, domains, and attributes. Further, it defines multiplicity constraints, specialization between entity types, and key attributes. There are several approaches to provide meta models for ERD but to our knowledge none defines the meaning of an entity in contrast to an entity type.

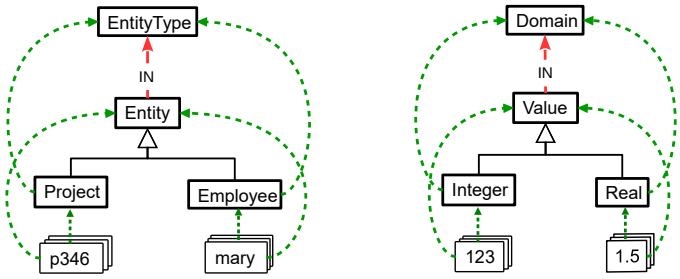

Fig. 3. Defining Entity and Value as most general instances

`Entity` in figure 3 is defined as most general instance of `EntityType`. As a consequence, the two entity types `Project` and `Employee` become subclasses of `Entity`, hence the instances `p346` and `mary` are both instances of `Entity`. This makes `Entity` a normal class that can be queried. In a symmetric way, `Value` is declared as most general instance of `Domain`. All values of the database can then be queried via the class `Value`. Even more, one can include a constraint that the two classes `Entity` and `Value` are disjoint. This is an implicit assumption in data modeling, which now becomes explicit.

The example can be pushed further in figure 4 by applying it to links, such as the the attribution `property` of `EntityType`. Links are first class objects in Telos and can also be subjected to the new construct for most general instances.

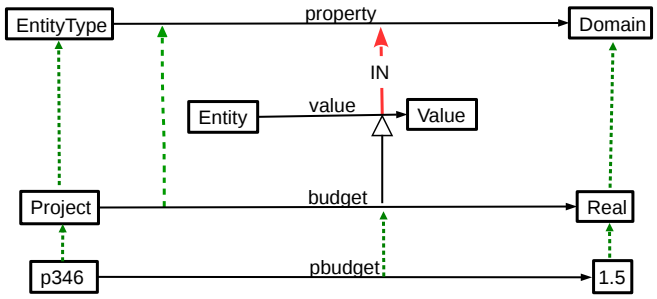

Fig. 4. Entity properties as most general instances

The link `value` of `Entity` is the most general instance of the link `property` of `EntityType`. The latter defines that entity types can have describing properties, such as the budget of projects. The most general instance `value` subsumes all data level links such as the `pbudget` link. We can thus use the `value` link to query all entities that have a certain property, regardless of the entity type! This allows for schema-less querying.

3.2 Ontological use of most general instances

We now turn to the running example to discuss the ontological use of most general instances for multi-level modeling. The essential idea of multi-level modeling is to define properties of objects and classes (clabjects) at the right level of abstraction in order to avoid redundancy and accidental complexity [2]. Existing approaches rely on potencies on links of clabjects, which are natural numbers specifying how many times the clabject has to be instantiated to reach the most specific incarnation of the clabject or link. For example, the link `property` in figure 4 would have the potency 2 since we reach after two instantiations to a link like `pbudget`, which me may classify as a fact that can not be instantiated further.

DeepTelos has no potencies at all and thus we need to show that it can be used for multi-level modeling. The replacement of potenticies are hierarchies of most general instances. In the running example, the central hierarchy is formed by `ProductCategory`, `ProductModel`, and `Product`. In OMG terms, `ProductCategory` would be a M3-level class (meta-metaclass), `ProductModel` would be a M2 class (metaclass), and `Product` be an M1 class (simple class).

Figure 5 shows the main chain of most general instances in the running example. The specializations to the MGI `ProductModel` are derived by axiom (5) from the instantiation facts

```
In(CarModel,ProductCategory)
In(PhoneModel,ProductCategory)
```

By declaring `Porsche911` as instance of `CarModel` it also becomes an instance of the MGI `ProductModel` via axiom (3). This again matches axiom (5) and makes

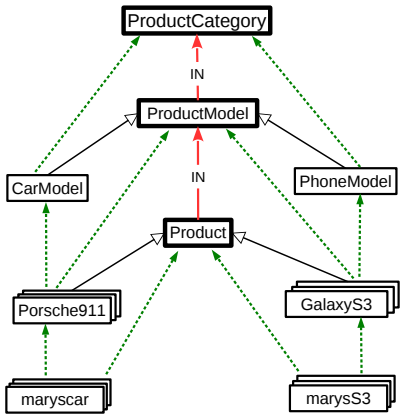

Fig. 5. The product hierarchy as chain of most general instances

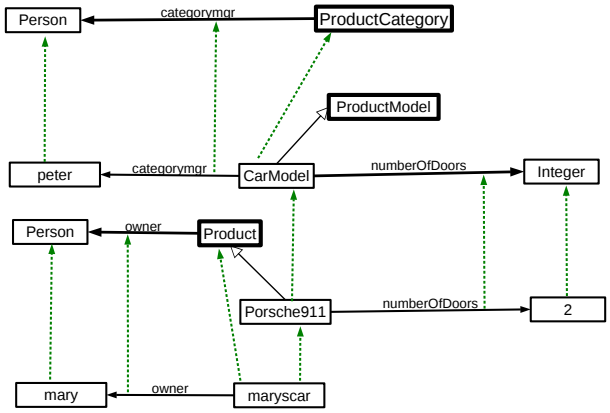

Fig. 6. Multi-level attributes for the product hierarchy

`Porsche911` a specialization of the MGI `Product`. Finally, `maryscar` is an instance of `Porsche911` and via axiom (3) an instance of `Product`. The right-hand side on phones works analogously. The MGI chain triggers the two axioms (3) and (5) and results in a set of derived instantiations to the MGI clabjects. These instantiations now allow to define the clabject properties of the running example in a way that does not violate the existing axioms of Telos, in particular the synchronous instantiation of the source and target of a proposition, axiom (4).

Figure 6 shows the use of the product hierarchy to model the properties of the running example. Product categories can have category managers. So, here `peter` is assigned as category manager of `CarModel`. Car models have a number of doors, here `Porsche911` has 2 doors. And finally, products have owners, e.g. `mary` owns

`maryscar`. The product hierarchy chain of figure 5 replaces the potencies at the expense of having multiple proxies of the 'product' concept, i.e. `ProductCategory` for potency 3, `ProductModel` for potency 2, and `Product` for potency 1. The benefit of the proxies is that we now have meaningful names for the abstraction levels. Like with Telos instantiation, there is no limit on the number of abstraction levels and they can also be extended at any time.

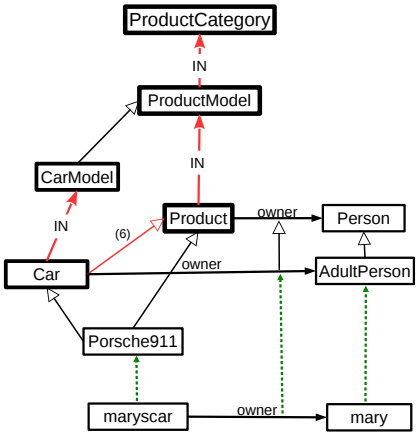

Fig. 7. Multiple MGI hierarchies

Several MGI hierarchies can co-exist and interact as shown in figure 7. Cars are defined as most general instances of `CarModel`. The 'owner' relation of `Product` is refined by `Car`, demanding that only adult persons may own a car (compare right side of figure 1). This case requires however that `Car` is a specialization of `Product`. We can ensure this by a second axiom for most general instances:

$$\forall\ c, d, m, n\ \ IN(m, c) \wedge IN(n, d) \wedge Isa(c, d) \Rightarrow Isa(m, n) \tag{6}$$

4 Implementation

The described approach is implemented and tested with several examples by extending ConceptBase with the required axioms. ConceptBase has Datalog as underlying computational engine. All user-defined formulas are compiled into an efficient Datalog program. For technical reasons, the *Isa* predicate is barred from occuring as conclusion of deductive rules. For this reason, the implementation defines a predicate `ISA` which behaves like the *Isa* predicate with respect to the inheritance of class membership, axiom (3). The code for the implementation uses a textual frame syntax. The new `IN` and `ISA` predicates are declared as attributes of `Proposition`:

```
Proposition with
  attribute
    ISA: Proposition; IN: Proposition
end
```

Subsequently, the new axioms are declared. Rule `mrule2` is equivalent to axiom (3) but now formulated for the user-defined `ISA` attribute of `Proposition`. Rules `mrule1` and `mrule3` are equivalent to axioms (5) and (6), respectively.

```
DeepTelosRules in Class with
  rule
    mrule1: $ forall m,x,c/Proposition
               (x in c) and (m IN c) ==> (x ISA m) $;
    mrule2: $ forall x,c,d/Proposition
               (c ISA d) and (x in c) ==> (x in d) $;
    mrule3: $ forall c,d,m,n/Proposition
              (m IN c) and (n IN d) and (c ISA d)  ==> (m ISA n) $
end
```

The product hierarchy is subsequently defined using the new `IN` relation. Note that the Telos *In* predicate is spelled '`in`' in the frame syntax. The source code includes in comments the potencies of clabject, attributes and relations. These comments serve for comparing DeepTelos with dual deep instantiation (DDI) [15,16], see discussion in Section 5.

```
ProductCategory with                    {* 3 *}
  attribute
    categoryMgr: Person                {* 1-1 *}
end
ProductModel  with
  IN c: ProductCategory
  attribute listPrice: Integer         {* 2-1 *}
end
Product  with
  IN c: ProductModel
  attribute owner: Person              {* 3-1 *}
end
CarModel in ProductCategory with        {* 2 *}
  categoryMgr c: peter                 {* 0-0 *}
  attribute numberOfDoors: Integer     {* 1-1 *}
end
Car with
  IN c: CarModel
  attribute mileage: Integer           {* 2-1 *}
end
Porsche911 in CarModel with             {* 1 *}
  numberOfDoors d: 2                   {* 0-0 *}
  listPrice p: 120000                  {* 0-0 *}
end
```

```
marysCar in Porsche911 with                {* 0 *}
  mileage m: 27000                       {* 0-0 *}
  owner o: mary                          {* 0-0 *}
end
Person end                                 {* 1 *}
peter in Person end                        {* 0 *}
mary in Person end                         {* 0 *}
```

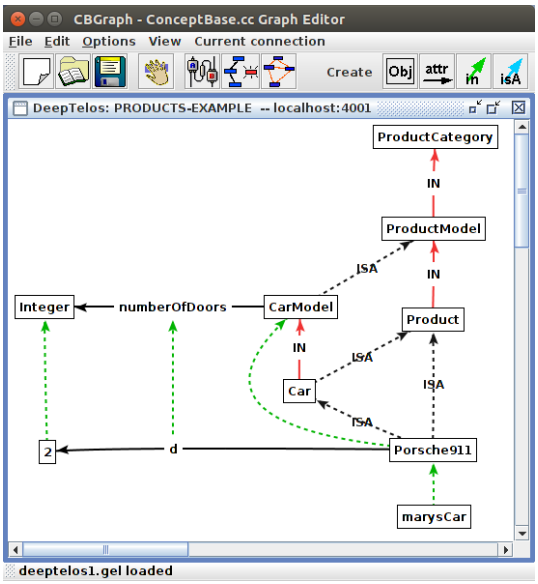

Fig. 8. ConceptBase screendump of the running example

Figure 8 shows a ConceptBase screendump of the running example. Not all attributes, instantiations, and specializations are shown in the screendump for sake of readability. ConceptBase uses a partial evaluation technique for formulas that contain predicates matching $In(x, c)$ with a variable c. Since the new axioms contain such predicates, the formula compiler produces efficient Datalog code for them. The implementation and examples can be downloaded under an open license from http://conceptbase.cc/deeptelos.

5 Discussion and Related Work

MGIs can be regarded as the inverse of Odell's *power types* [18]. Adding to a metaclass c an MGI m results in a base class m with a power type c. The contribution of DeepTelos is to fully integrate this construct in a metamodeling language and system which already comes with support for unbounded deep characterization via

'mediated' [20] properties and full support for metaclasses. This also sets DeepTelos apart from work on the powertype pattern [4] where the powertype role is played by a 'normal' class. For an insightful analysis of power types and their role in multi-level modeling see [3]. We take a perspective that is inverse to power types, because our starting point are metaclasses in Telos. With MGIs we add full support for deep characterization to Telos, making it possible that the metaclass introduces (via its MGI) a property that is directly instantiated by the individual (its instance-instance, i.e., potency 2) without an intermediate instantiation step at the class – this is similar to what Rossini [20] refers to as 'semantics of single-potency'.

DeepTelos is related to the metamodeling system VODAK [11], where a metaclass *c* comes with own-type, instance-type, and instance-instance-type, which are all specified together with the metaclass. In that way, VODAK supported deep characterization, but limited to two instantiation levels. The added value of DeepTelos is to have unbounded meta-levels. For example, `ProductCategory` (specifying own-values and instance-type) would typically be created together with its MGI `ProductModel` (specifying the instance-instance-type of `ProductCategory`) and its MGI's MGI `Product` (specifying the instance-instance-instance-type of `ProductCategory`). While DeepTelos is not restricted to a particular modeling methodology, this example gives a hint of how MGIs are applied by metamodelers.

DeepTelos is also related to our work on Dual Deep Instantiation [15,16]. In DDI, a class together with its MGI chain would be represented by a single clabject where each property has source and target potency. For example (see comments in the example in Section 4), `ProductCategory`, `ProductModel`, and `Product` would be represented together by a single clabject with potency 3, with property `listPrice` having source potency 2 and target potency 1.

Linguistic vs Ontological Instantiation: In multi-level modeling one often distinguishes two types of classification (or instantiation), namely linguistic classification and ontological classification [6]. It is often argued that these two kinds of instantiation are orthogonal to each other. In a first approach [15] to use O-Telos for multi-level modeling, we specified and implemented dual deep instantiation in ConceptBase. We followed the idea of separating linguistic instantiation (defined by the O-Telos axioms) from DDI's constructs for ontological instantiation (defined by roughly another 30 new axioms). The result was consistent and the axioms were free of redundancy, but it showed that this separation hinders the modeler from making use of Telos' metamodeling features together with DDI's multi-level modeling features, further, the combined set of axioms was relatively large and their execution in ConceptBase rather slow.

The DeepTelos approach, in contrast, preserves all the strengths of Telos and ConceptBase by not distinguishing linguistic and ontological instantiation. Interestingly, there seem to be different kinds or levels of linguistic instantiation: first, all model elements in a DeepTelos model are linguistic instances of `Proposition`, second, when modeling a modeling language like ERD (see Section 3.1) with linguistic classes `Entity` and `EntityType`, then some model elements, like `mary`, which is a linguistic instance of `Proposition` is also a linguistic instance of `Entity`. It seems that the former is a linguistic instantiation in DeepTelos and the second is a linguis-

tic instantiation in ERD, which is in turn modeled in DeepTelos. The two scenarios about the linguistic use (see Section 3.1) and ontological use (see Section 3.2) of MGIs can be combined and create no inconsistency because the two hierarchies are separate. The combination allows to query for products that are also entities, or for the identifier (key attribute) of a given product.

One may ask whether MGI hierarchies are model-specific or universal. This cannot be answered from the axiomatic standpoint used in this paper. The 'ontological' MGI hierarchy for products has more of a model-specific flavor, since there are these 3 levels for product categories, product models, and actual products. This is a deliberate choice of the modeler. The 'linguistic' example for entity types and entities has a more universal nature because of the set-theoretic semantics of entity types as defined by the creator of the entity relationship model. A closer investigation of this question is subject to future research.

Formalization of DeepTelos: The DeepTelos axiomatization requires virtually only a single additional axiom (5) for Telos to realize an environment for multi-level modeling. The main axiom (5) is surprisingly simple and similar to the class membership axiom (3) of Telos. Both axioms closely interact with each other: the specializations derived by axiom (5) feed into the condition of axiom (3) to derive new instantiations. These instantiations then feed again into the condition of axiom (5). A second axiom (6) was added to ensure consistency of hierarchies where subtypes have MGIs. For example, in Fig. 7, `CarModel` with MGI `Car` is a specialization of `ProductModel` with MGI `Product`; axiom (6) then derives that `Car` is a specialization of `Product`.

As an alternative to MGIs, we also investigated extending Telos with singleton hierarchies (which we referred to internally as *most specific classes*). This construct did work for part of the example but finally clashed with a naming axiom in Telos that forbids that objects have multiple attributes/relations with the same label. The singleton approach apparently was also heavier in terms of required additional axioms, so we abandoned it. Both approaches share the idea to have proxy objects at the required level to replace the usual potencies of multi-level modeling approaches.

More complex MGI networks: We provided only examples of MGI chains, where a class has at most one MGI and an object is MGI of at most one class. This is the typical case, but it is worth discussing how more complicated MGI networks operate. First, consider the case that a class c has two MGIs m_1 and m_2. Axiom (5) will then derive the same subclasses for both m_1 and m_2. If the two MGIs are abstract classes, then they shall always have the same set of instances. One may then argue that they are redundant and multiple MGIs of the same class should be excluded. Even if the two MGIs had different attributes, then every instance of the first MGI would also be an instance of the other one. Hence, they always can use all attributes. As a consequence, we may want to forbid multiple MGIs of the same class c.

A second case is that an object m is the most general instance of two different classes c_1 and c_2. In this case, it shall have the instances of c_1 and the instances of c_2 as subclasses. This is useful when having different classifications for the same kind of objects. In our running example instead of a single `ProductCategory` class there may be multiple categorizations of product models, e.g., `SalesProductCategory`

and `LogisticsProductCategory` which have different instances, e.g., `LuxuryCar` and `GoodsForResale`, respectively, but `ProductModel` as most general instance.

A network of MGI relations could be cyclic, e.g. $IN(m_1, m_2), IN(m_2, m_1)$. The instances of the one would then be subclasses of the other. We see no real-world application of such a pattern but did not forbid it in the axioms. We rather followed a minimalistic approach and leave additional constraints open for the reader.

6 Conclusions

We presented a specification of multi-level modeling within Telos that exploits the existence of the Telos axioms to keep it very simple, yet consistent with the Telos specification of instantiation and specialization. We further presented the implementation of DeepTelos in ConceptBase. Since ConceptBase is a highly optimized implementation of Telos, the extended language DeepTelos also has an efficient implementation. This implementation is available under an open license and can be downloaded to be applied and extended for own research and experimentation. Future work includes the following:

- DeepTelos should be tested with more example models from other multi-level modeling approaches to see its precise limitations.
- The restriction on the *Isa* predicate implementation in ConceptBase should be lifted to avoid the detour via the self-defined `ISA` relation.
- Additional axioms, e.g. forbidding the cyclicity of *IN* should be investigated.
- The introduction of additional axioms and relations can be guided by Carvalho's work [3] on ontological foundations of multi-level modeling.
- The linguistic and ontological use of the new construct of most general instances deserves more research with respect to utility and consistency.

References

1. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: UML 2001. LNCS, vol. 2185, pp. 19–33. Springer (2001), http://dx.doi.org/10.1007/3-540-45441-1_3
2. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software and System Modeling 7(3), 345–359 (2008), http://dx.doi.org/10.1007/s10270-007-0061-0
3. 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: ER 2015, LNCS, vol. 9381, pp. 119–133. Springer (2015), http://dx.doi.org/10.1007/978-3-319-25264-3_9
4. Gonzalez-Perez, C., Henderson-Sellers, B.: A powertype-based metamodelling framework. Software and System Modeling 5(1), 72–90 (2006), http://dx.doi.org/10.1007/s10270-005-0099-9
5. Greenspan, S.J., Mylopoulos, J., Borgida, A.: On formal requirements modeling languages: RML revisited. In: ICSE 1994. pp. 135–147. IEEE Computer Society / ACM Press (1994), http://portal.acm.org/citation.cfm?id=257734.257754

6. Gutheil, M., Kennel, B., Atkinson, C.: A Systematic Approach to Connectors in a Multi-level Modeling Environment. In: MoDELS. LNCS, vol. 5301. Springer (2008), http://dx.doi.org/10.1007/978-3-540-87875-9_58
7. Henderson-Sellers, B., Gonzalez-Perez, C.: The rationale of powertype-based metamodelling to underpin software development methodologies. In: APCCM 2005. pp. 7–16. Australian Computer Society (2005), http://crpit.com/confpapers/CRPITV43HendersonSellers.pdf
8. Jarke, M., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M., Eherer, S.: ConceptBase - a deductive object base for meta data management. J. Intell. Inf. Syst. 4(2), 167–192 (1995), http://dx.doi.org/10.1007/BF00961873
9. Jeusfeld, M.A.: Complete list of O-Telos axioms (2005), online: http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/d1228997/O-Telos-Axioms.pdf
10. Jeusfeld, M.A., Jarke, M., Mylopoulos, J. (eds.): Metamodeling for Method Engineering. Cooperative information systems, Cambridge, Mass. MIT Press (2009)
11. Klas, W., Schrefl, M.: Metaclasses and Their Application - Data Model Tailoring and Database Integration. Springer (1995), http://dx.doi.org/10.1007/BFb0027185
12. de Lara, J., Guerra, E.: Deep meta-modelling with MetaDepth. In: TOOLS (48). LNCS, vol. 6141, pp. 1–20. Springer (2010), http://dx.doi.org/10.1007/978-3-642-13953-6_1
13. de Lara, J., Guerra, E., Cuadrado, J.S.: When and how to use multilevel modelling. ACM Trans. Softw. Eng. Methodol. 24(2), 12:1–12:46 (2014), http://doi.acm.org/10.1145/2685615
14. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: Representing knowledge about information systems. ACM Trans. Inf. Syst. 8(4), 325–362 (1990), http://doi.acm.org/10.1145/102675.102676
15. Neumayr, B., Jeusfeld, M.A., Schrefl, M., Schütz, C.: Dual deep instantiation and its ConceptBase implementation. In: CAiSE 2014. LNCS, vol. 8484, pp. 503–517. Springer (2014), http://dx.doi.org/10.1007/978-3-319-07881-6_34
16. Neumayr, B., Schuetz, C.G., Jeusfeld, M.A., Schrefl, M.: Dual deep modeling: multi-level modeling with dual potencies and its formalization in F-Logic. Software & Systems Modeling pp. 1–36 (2016), http://dx.doi.org/10.1007/s10270-016-0519-z
17. Nissen, H.W., Jeusfeld, M.A., Jarke, M., Zemanek, G.V., Huber, H.: Managing multiple requirements perspectives with metamodels. IEEE Software 13(2), 37–48 (1996), http://dx.doi.org/10.1109/52.506461
18. Odell, J.J.: Advanced object-oriented analysis and design using UML, chap. Power types, pp. 23–32. Cambridge University Press (1998)
19. Pirotte, A., Zimányi, E., Massart, D., Yakusheva, T.: Materialization: A Powerful and Ubiquitous Abstraction Pattern. In: VLDB. pp. 630–641 (1994), http://www.vldb.org/conf/1994/P630.PDF
20. Rossini, A., de Lara, J., Guerra, E., Rutle, A., Wolter, U.: A formalisation of deep metamodelling. Formal Aspects of Computing 26(6), 1115–1152 (2014), http://dx.doi.org/10.1007/s00165-014-0307-x