Multi-level Modeling without Classical Modeling Facilities

Ferenc A. Somogyi[1][a], Zoltán Theisz[2][b], Sándor Bácsi[1][c], Gergely Mezei[1][d] and Dániel Palatinszky[1][e]

[1]*Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Muegyetem rkp. 3., Budapest, Hungary*

[2]*Evopro Systems Engineering Ltd., Budapest, Hungary*

{*somogyi.ferenc, sandor.bacsi, gergely.mezei, daniel.palatinszky*}*@aut.bme.hu, zoltan.theisz@evopro-group.com*

Keywords: Multi-level Modeling, Multi-layer Modeling, Potency Notion.

Abstract: Multi-level modeling is a modeling paradigm that has been becoming more and more popular in recent years. The ultimate goal of multi-level modeling is to reduce accidental complexity that may arise from modeling methodologies that are not suitable to describe multi-level models. However, currently, most multi-level modeling approaches are dependent on classical modeling facilities, like OMG's four-level modeling architecture. The potency notion is one of the most widely-used enablers of multi-level modeling that governs the depth model elements can be instantiated at. In this paper, we propose an alternative implementation to the first incarnation of the potency notion, the so-called classic potency notion. Its multi-level nature is mapped into our multi-layer modeling framework, which liberates it from direct dependence on classical modeling facilities. We examine the proposed mapping in details and also demonstrate it via a simple case study. This work is aimed as the first step towards a generic multi-level modeling framework where researchers can freely experiment with novel multi-level modeling ideas and can share and compare their results without worrying about any sorts of tool dependence.

1 INTRODUCTION

Multi-level modeling (Atkinson and Kühne, 2001) is a well-established paradigm within the realm of modern modeling approaches based on the initial core ideas of OMG's Model Driven Architecture (MDA) (MDA, 2001). Multi-level modeling had been originally conceived as a technical extension to go beyond the strictly preset four-level meta-model layering prescribed by OMG's MDA blueprint and related technical documentation. OMG does not seem to fully embrace multi-level modeling and give its official recognition to acknowledge full membership within the standardized modeling facilities. Thus, it has mostly remained and has been developed inside the modeling research community. Moreover, in recent years, it has enjoyed a renaissance by the rediscovery of some of its most successful techniques, such as the potency notion (Atkinson and Kühne, 2001; Kühne*, 2018). Both theoretical and practical interest in applying multi-level modeling to real case studies has been growing not only within, but also beyond the modeling research community. Therefore, we believe that the potential of thorough analysis in an effective research playground for multi-level modeling and its foundations is beneficial to the academic research community and – in the longer term – for the industry as well.

Our motivation of writing this paper has been mostly influenced by our interest to look for alternatives of doing multi-level modeling in Orthogonal Classification Architecture (OCA) (de Lara et al., 2014). In OCA, entites have two orthogonal classifical dimensions: the ontological dimension defined by the modeled domain, and the linguistic dimension defined by the modeling structure. In our understanding, multi-level modeling in OCA is mainly a modeling paradigm that is supposed to relax the constraints of OMG's four-level modeling stack by the introduction of additional modeling levels. Hence, as an add-on supplement to legacy meta-modeling, multi-level modeling requires every facility of OMG's modeling infrastructure: (1) object-oriented modeling concepts like inheritance and (2) standardized tool sup-

[a] https://orcid.org/0000-0001-5491-4412
[b] https://orcid.org/0000-0002-0222-867X
[c] https://orcid.org/0000-0002-4814-6979
[d] https://orcid.org/0000-0001-9464-7128
[e] https://orcid.org/0000-0002-6168-3963

port for OMG-style modeling with predefined meta-levels. Therefore, OCA can be regarded as a peculiarly constructed co-architecture to OMG's MDA stack. Namely, OCA integrates multi-level modeling into the standard linguistic meta-level setup of OMG's MDA framework by the presumption of the so-called ontological instantiation concept. The main reason of OCA's success is the fact that its ontological instantiation assumes or prescribes no implicit semantics in OMG's MDA design, which positions it clearly against the linguistic instantiation that does require it. Hence, state-of-the-art MDA tools may be capable of supporting OCA out-of-the-box. Nevertheless, the orthogonal coupling of the two instantiations also has consequences: on the positive side, OCA can easily provide smooth integration of multi-level modeling into the standard modeling framework; on the negative side, OCA's particular interpretation of multi-level modeling is not semantically defined within OMG's MDA meta-modeling framework. Thus, the semantics of ontological instantiation must be encoded, mainly in an ad-hoc manner, inside the various modeling tools. Therefore, OCA's multi-level modeling is by its very nature tool dependent without offering any encompassing theory to reunite the particular implementation flavours of similar or same multi-level modeling concepts. A good comparison to observe the divergence in semantics of similar concepts in ontological instantiation is to be found in (Gerbig et al., 2016).

By analyzing some of the core concepts of OCA's multi-level modeling, it may seem almost as obvious that the potency notion can be regarded as the principal holder of semantics for the definition of ontological levels via deep instantiation. Indeed, the potency notion introduces for example an abstract counter that is to be attached to the core OMG meta-modeling entities, that is, to nodes, fields, and edges. Thus, the semantics of the potency notion defines thereby the required layering of ontological levels. Based on the orthogonal design, OCA only had to rely on OMG's linguistic instantiation for its compliant functionality within MDA, which ultimately resulted in a theoretical-practical hybrid solution. However, it also meant that there was no purely abstract implementation of the potency notion semantics without it being bundled to legacy modeling tools or languages.

In this paper, we intend to report on an alternative implementation of the original (classic) potency notion (Atkinson and Kühne, 2001) specification within our multi-layer modeling framework not based on OCA. The framework only plays the role of an executable playground for specifying the logic of the potency notion without following a rigid specification hard-coded in a tool. Therefore, we think that our re-implementation of the classic potency notion clearly demonstrates how legacy tool-dependence can be avoided. Namely, we relied only on our neutral interpretation of deep instantiation, on its abstract mathematical level, our modeling tool is based on. Although the existence of multiple modeling layers is supported by our formalism, the semantics of those modeling layers is neutral from the point of view of the potency notion since it differs from that of the levels potency notion does require. Thus, the potency notion had to be encoded (modeled) through an explicit semantics mapping between the underlying deep instantiation by multi-layering and the specified constraints defined on the functionality of the potency notion. Although there is no explicitly required dependency on classical modeling facilities in our proposal, for the reason of practical model validation a framework implementation has also been created[1].

The paper is structured as follows. In Section 2 we discuss the concept and brief history of the potency notion. We also introduce our multi-layer modeling framework. Section 3 contains an explicit specification of the classic potency notion. Section 4 presents the mapping of multi-level concepts into our multi-layer modeling framework in reasonable detail, and also includes a case study. Finally, Section 5 concludes the paper by highlighting future work in this research field.

2 BACKGROUND AND RELATED WORK

2.1 The Potency Notion

The main goal of multi-level modeling is to reduce accidental complexity (Atkinson and Kühne, 2008) by using an unlimited number of meta-levels to assign the right amount of abstraction details to every level. Accidental complexity refers to parts of the solution that are needed to express its multi-level nature, instead of describing the domain in question. Using the Item Description pattern to describe multiple domain levels in object-oriented languages is an example of such accidental complexity (Atkinson and Kühne, 2008). Levels have been present for a long time in modeling, dating back to the Unified Modeling Language (UML) (Fowler and Scott, 2000). Kühne argues that well-defined levels "safeguard against ill-formed models" (Kühne, 2018), which means that it is easier to create valid, well-formed models. Atkinson

[1] https://www.dmla.hu

and Kühne compare this concept to strongly-typed programming languages, while approaches that "ignore the fact that there are levels" are akin to weakly-typed languages (Atkinson et al., 2014). They call the former *level-adjuvant*, and the latter *level-blind* approaches.

The basic enablers of most level-adjuvant multi-level modeling approaches are i) the notion of clabjects (Atkinson and Kühne, 2000) and ii) the so-called potency notion (Atkinson and Kühne, 2001). Clabjects are entities that have a class and object facet at the same time, thus, they can behave as either of the two. This makes it possible to instantiate elements at any level in the model. The potency notion is a mechanism responsible for governing deep instantiation, which is the mechanism of instantiation across multiple levels. In the context of this paper, we are mostly focusing in the original version of potency (Atkinson and Kühne, 2001), often referred to as "classic potency" (Kühne*, 2018). Our long-term goal is to implement all the related concepts of the potency notion in order to harmonize the different solutions by identifying their practical strengths and weaknesses.

In the interpretation of the classic potency notion, the potency value is an integer number given to every model element (nodes, edges, and fields) that defines the depth to which a model element can be instantiated (Atkinson and Kühne, 2001). Every time a model element is instantiated, its potency value (and its level) is decreased by one. The potency value of a model element must always be less or equal than its level. This allows the creation of model elements that reach their fully concretized state (a potency value of 0) at a level higher than 0. Elements with a potency value of 0 cannot be instantiated further. Fields (features of nodes) are differentiated based on whether they can only get a value when their potency reaches 0 (called single fields), or they get a value every time their potency decreases (called dual fields).

Inheritance is an important concept in both the object-oriented and modeling world. While the paper describing the classic potency notion (Atkinson and Kühne, 2001) does not elaborate on inheritance much, we felt this was too important a concept to be left out of our initial work. Therefore, we took inspiration from later work in this regard (Kühne*, 2018). This extended classic potency notion (extended with inheritance) is explicitly specified in Section 3.

Using the two main concepts of clabjects and the potency notion, model elements can receive features from more than one level above them, which is referred to as deep instantiation (Atkinson and Kühne, 2001). This stands in contrast with the shallow instantiation mechanism of UML and other MOF-based architectures (MOF, 2005), as instantiation in this case only covers two modeling levels at a time. Among the main motivations of multi-level modeling are i) the better expressive power of models, and ii) lessened complexity of models by eliminating accidental complexity.

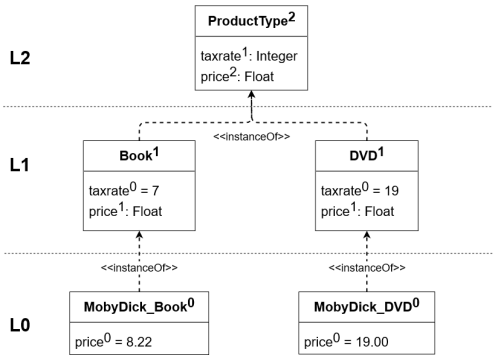

Figure 1: Online store domain as a multi-level model.

Figure 1 depicts a simple multi-level model using the classic potency notion. The example was taken from related work (Kühne and Schreiber, 2007). The domain in question is an online store that sells different products: books and DVD-s. Product types have a tax rate, and every instance of a product type has a price. In this online store domain, we can see how the potency notion governs deep instantiation. The *taxRate* feature of the clabject *ProductType* receives a value one level below (in clabjects *Book* and *DVD*), while the *price* feature receives it two levels later.

The concept of clabjects and the potency notion were introduced in many approaches, for example, in DeepJava - an object-oriented language extended to multiple levels - in order to support multi-level programming (Kühne and Schreiber, 2007). This is one of the earliest applications of multi-level concepts. DeepJava seemed to be a feasible solution to multi-level programming, although being based on an object-oriented language, there was still some accidental complexity left that it aimed to avoid. There were also some pitfalls with the original potency notion that later variations aimed to improve upon. Various multi-level modeling approaches emerged in recent years (Atkinson and Gerbig, 2016; de Lara and Guerra, 2010; Clark and Willans, 2012), and the potency notion received many alterations, variations and interpretations over time (Neumayr et al., 2018; Frank, 2014; Atkinson et al., 2012). These variations were discussed in related work (Atkinson et al., 2014; Kühne*, 2018; Kühne, 2018), and is not the scope of this paper to elaborate upon.

2.2 Dynamic Multi-layer Algebra

In this section, we summarize the most important features of Dynamic Multi-Layer Algebra (DMLA) (Urbán et al., 2018; Urbán et al., 2017). DMLA is a multi-layer modeling framework that consists of two main parts: (i) the Core containing the formal definition of modeling structures and its management functions; (ii) the Bootstrap having a set of essential reusable entities of any modeled domains in DMLA.

The Core contains the formal mathematical definition how each model entity is defined as a 4-tuple (unique ID, meta-reference, attributes and concrete values). Beyond giving the formal definition of the tuples, the Core also declares some primitive functions that directly manipulate the model, like adding new model entities or select existing ones. These mathematical constructs together define the Core of DMLA, which is interpreted over an Abstract State Machine (ASM) (Börger and Stärk, 2003). The Bootstrap (Mezei et al., 2019) is the initially defined set of tuples needed for meta-modeling constructs and built-in model elements.

Instantiation in DMLA means gradual constraining (refinement). Whenever a model entity claims another entity as its meta, the framework automatically validates if there is indeed a valid instantiation between the two entities. The semantics of valid instantiation are modeled by the Bootstrap entities. When an entity is being instantiated, one can decide which of the slots are being instantiated and which are merely cloned, that is, being copied to the instance without any modifications. Besides gradually narrowing the constraints imposed on a slot, DMLA also enables the division of a slot into several instances, thus fragmenting a general concept into several, more specific ones. Moreover, it is also possible to omit a slot completely during the instantiation if it does not violate its cardinality constraint. Similarly to the relation between an entity and its meta-entity, each slot originates from a meta-slot defining the constraints it must take into consideration. These include, but are not limited to type and cardinality constraints.

The most important entity, *ComplexEntity*, has a slot called *Children*. The cardinality of the slot *Children* allows any number of instances (0..*) of any practically available type. Since all slots in DMLA must have a meta, it is not possible to add new features to an entity, unless the meta-entity has an appropriate meta-slot. The *Children* slot allows us to extend entities by adding new slots. If we omit the slot, we can deny any further introduction of slots into the instances of the given entity.

3 SPECIFICATION OF THE CLASSIC POTENCY NOTION

In this section, we present the specification of the classic potency notion as discussed in Section 2. In some cases, where the original work (Atkinson and Kühne, 2001) was not clear enough (like in the case of handling inheritance), we borrowed ideas from later work (Kühne and Schreiber, 2007; Kühne*, 2018). These differences can be found when comparing Figures 1 and 3, and are explained later.

3.1 Level-related Requirements

L1 Levels serve as a logical separation between different model elements (Kühne, 2018). Every model element (nodes, edges and fields) is assigned a level value.

$$\forall ME \in ELEMENTS : \exists ME.l \tag{1}$$

L2 A level value is always greater than or equal to 0.

$$\forall ME \in ELEMENTS : ME.l \geq 0 \tag{2}$$

L3 The level of a field in a node is always equal to the level of the node itself.

$$\forall F_N \in fields(N) : F_N.l = N.l \tag{3}$$

L4 When an instantiation occurs, the level of the model element decreases by 1. This means that the level of an instance is always less than the level of its meta element by 1.

$$meta(ME_1, ME_2) : ME_1.l = ME_2.l - 1 \tag{4}$$

L5 Cross-level references are not allowed. A cross-level reference is an edge that runs between two nodes that are on different levels.

$$\forall E \in EDGES : E.N1.l = E.N2.l = E.l \tag{5}$$

3.2 Potency-related Requirements

P1 Potency governs the depth a model element can be instantiated. Every model element (nodes, edges and fields) is assigned a potency value.

$$\forall ME \in ELEMENTS : \exists ME.p \tag{6}$$

P2 A potency value is always greater than or equal to 0.

$$\forall ME \in ELEMENTS : ME.p \geq 0 \tag{7}$$

P3 The potency of a field in a node is always less than or equal to the potency of the node except if the node is an abstract node.

$$\forall F_N \in fields(N) : F_N.p \leq N.p; \quad N \in NODES - ABSTRACT \tag{8}$$

P4 When an instantiation occurs, the potency of the model element decreases by 1. This means that the potency of an instance is always less than the potency of its meta element by 1.

$$meta(ME_1, ME_2) : ME_1.p = ME_2.p - 1 \quad (9)$$

P5 If the potency of a model element is 0, then it cannot be instantiated further.

$$ME_1.p = 0 : \nexists ME_2 : meta(ME_2, ME_1) \quad (10)$$

P6 Abstract nodes have a potency value of 0. Therefore, they cannot be instantiated, they can only be inherited from.

$$N_A \in ABSTRACT : N_A.p = 0 \quad (11)$$

P7 Abstract nodes can only have fields with a potency value greater than 0.

$$\forall F_N \in fields(N_A) : F_N.p > 0; N_A \in ABSTRACT \quad (12)$$

P8 Single fields only get a value at potency 0. Dual fields get a value every time their potency decreases.

$$F.p \neq 0, F \in SINGLE : \nexists F.val \text{ and } F.p \neq 0, F \in DUAL : \exists F.val \quad (13)$$

P9 When a field gets a potency value of 0, it is omitted from the instance node in the next instantiation step.

$$\forall F \in fields(M), meta(I, M) : F \notin fields(I); F.p = 0, I, M \in NODES \quad (14)$$

P10 The potency of a model element is always less than or equal to its level.

$$\forall ME : ME.p \leq ME.l \quad (15)$$

P11 Inheritance is a special edge that always has a potency of 0 and connects two nodes.

$$inh(N_1, N_2).p = 0; N_1, N_2 \in NODES \quad (16)$$

P12 When inheriting from a node, the meta node of the descendant has to be the meta node of the parent.

$$inh(D, P), meta(D, M) : meta(P, M); D, P, M \in NODES \quad (17)$$

4 MAPPING OF THE CLASSIC POTENCY NOTION

In this section, we discuss the mapping of the classic potency notion in our multi-layer modeling framework, DMLA. First, we present the architecture of the mapping, namely, where the mapping takes place in the current architecture of DMLA. Next, we discuss how each of the requirements described in Section 3 are satisfied, giving more details on cases where the mapping is not trivial. Finally, we demonstrate the mapping of the classic potency notion through a simple case study. It is important to emphasize that our mapping solution is devoid of any tool-imposed limitations since we only rely on the mathematically backed up validation mechanism of DMLA in order to implement the semantics of the classic potency notion.

4.1 Mapping Architecture

There were two theoretical options in DMLA on how to map the classic potency notion onto it: i) by introducing a new Bootstrap and build up a new set of entities from the ground up, or ii) by introducing a new stratum of entities that is responsible for the mapping between DMLA's layer and the potency notion's level-based semantics. After thorough deliberation, we decided on the second option, since by reusing the entities already defined in the standard Bootstrap, it seemed sufficient to only implement the mapping between the semantics. Thus, the model validation encoded in DMLA entities can also verify the interpretation logic of the mapping itself. The mapping stratum lies directly below the Core and the Bootstrap, which means that it is only dependent on them. Domain models will be able to easily apply the classic potency notion in DMLA as they are positioned below the mapping stratum. Thus, our choice of mapping implementation allowed us to create an effective mapping infrastructure that can be further extended by other multi-level modeling concepts.

4.2 Mapping Details

In the following, we describe the mapping of the classic potency notion based on the specification described in Section 3. Figure 2 visually depicts the mapping from the concepts of the potency notion to their implementation in DMLA. The top of the figure depicts the elements of the classic potency notion, while the bottom shows the structure of the mapping in DMLA. The slots are shown embedded into the entities. Meta-slot relationships are represented with

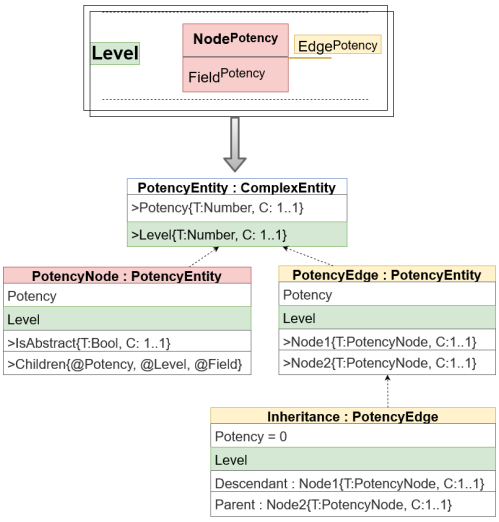

Figure 2: Mapping multi-level concepts in DMLA.

Slot:MetaSlot labels. The different constraints (type, cardinality, potency, level, and field) are expressed between curly brackets. As we can see, the basic concepts of nodes and edges are mapped to separate entities, while fields are mapped to slots in DMLA. While the internal technicalities of DMLA's framework are out of scope of this paper, we believe that we should not totally disregard them for the sake of better understanding of the mapping logic. Thus, we describe the main concepts of the mapping, but we omit low level details, such as the syntax of DMLA's operation language. Nevertheless, the complete DMLA solution can be found on our webpage[2]. In the following, we explain the mapping, referencing Section 3 regarding the satisfied requirements:

- **PotencyEntity.** The entry point to the mapping layer, nodes and edges are instantiated from this entity. Contains level and potency values as slots, which are mandatory and is of a simple number type. This partially satisfied requirements [L1, P1]. Due to the nature of validation in DMLA, the validation logic defined in this entity validates every entity instantiated from it, including nodes and edges. The validation logic checks basic requirements regarding the level and potency values, i.e., the aforementioned values must be greater than 0, or that they decrease by 1 when instantiating, and so on [L2, L4, P2, P4, P5, P10]. In the validation code of the *PotencyEntity*, we also check if there is a valid instantiation between two entities regarding their potency values. Namely, we cannot instantiate from a node with a potency value of 0 (such as abstract nodes), and that fields with a potency value of 0 must be omitted from the instance [P6, P9].
- **PotencyNode.** Maps the concept of nodes from the classic potency notion. The *Children* slot is the new entry point for defining fields in domain models. Fields are annotated with potency, level, and the nature of the field (single or dual). Abstract nodes are marked by the *IsAbstract* flag, and they are checked to have a potency of 0 in the validation logic [P6]. The validation logic checks i) if an abstract node only has fields with potency greater than 0 [P7], and ii) the relationship between field and node level and potency [L3, P3].
- **PotencyEdge.** Edges are modeled as entities in our mapping, since we can apply level and potency to the edges, and we can more easily manage cross-level references. Also, it is possible to model special edges like inheritance more accurately this way, using the flexible validation logic of DMLA. The *PotencyEdge* entity has two connected nodes as slots, and validates against cross-level references (checking the level values of the edge and its two connected nodes) [L5]. It also checks if the connected nodes conform to the connected nodes in the meta edge, if there are any.
- **Inheritance** is a special edge that has a potency of 0 [P11]. Inheritance is instantiated from *PotencyEdge*, thus, it behaves like an edge, but has special constraints. The validation checks that every field in the parent must be present in the descendant, and that the descendant must instantiate the meta of the parent [P12], thus simulating the concept of inheritance.
- **PotencyConstraint.** A potency value is assigned to fields in the form of a constraint [P1]. The validation logic checks the basic rules for the potency value, similarly to *PotencyEntity* [P2, P4, P10]. We also validate that the field gets an assigned value when it reaches potency 0 [P8].
- **LevelConstraint.** Behaves similarly to *PotencyConstraint*, assigning a level value to fields [L1]. Validation is also similar, checking the basic rules of level values [L2, L4].
- **FieldConstraint** contains a simple flag that can be used to mark a field as single or dual [P8]. The validation is responsible for checking the assigned values as described by the classic potency notion [P8]. It also ensures that once the nature of a field is defined, it can never be overwritten. The constraints from Figure 2 can be found on the *Children* field in *PotencyNode*, from which every new field is created.

[2]https://www.dmla.hu

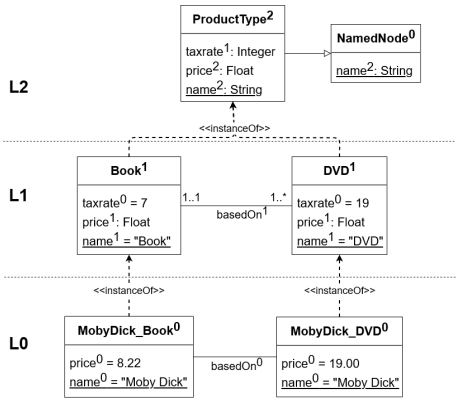

Figure 3: The extended online store model.

4.3 Case Study

The case study was inspired by examples found in related work (Kühne and Schreiber, 2007). Figure 3 depicts the online store domain previously seen in Figure 1, extended with new concepts, which are as follows. Every *ProductType* instance (and thier instances, and so on) have a name. This means that *Book* has a defining name (”Book”), while book instances also have their own name (like ”Moby Dick”). To realize this, *ProductTypes* inherit from *NamedNode*, which is an abstract node with a potency value of 0. Its *name* field is a dual field, which means that it must receive a new value every time its potency decreases. Please note that while *ProductType* could be an abstract clabject, we chose to remain faithful to the original example, and left it as non-abstract. The other extension is that every DVD is based on a book, which is modeled as an association with a role name. Every field other than *name* is single, which means that they only receive a value when their potency reaches 0. On the middle level, we have the *ProductType* instances: *Book* and *DVD*. There is an association between the two (*basedOn*), which has a potency value of 1, meaning that it can be instantiated one level below. On the lowest level, we have the *Book* and *DVD* instances: *MobyDick_DVD* is based on *MobyDick_Book*.

Figure 4 depicts the mapped online store domain in DMLA. The most significant differences are perhaps the modeled edges which appear as entities (like *BasedOn*). The notation on the figure is deliberate and is aimed at highlighting the differences compared to classical modeling facilities. For example, since there are no explicit levels in DMLA, the levels appear as they are present in the mapping, namely, as slots (fields) in the entities. That is a DMLA technicality and it has nothing to do with the semantics of the levels. Constraints on the slots are marked be-

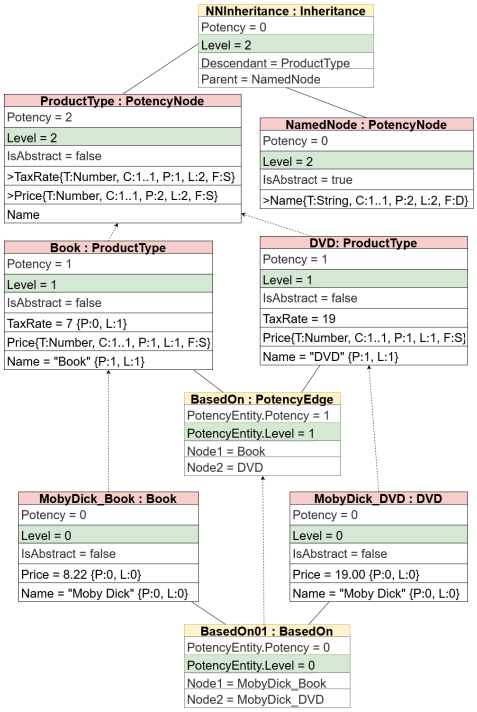

Figure 4: Online store domain mapped in DMLA.

tween curly braces, similarly to Figure 4. For example, the *TaxRate* slot in the *ProductType* node has a type of *Number*, a cardinality of 1..1, a potency of 1, a level of 2, and it is a single field (”S” means single, ”D” means dual). Please note that the validation logic does not appear in the figure, as it is not part of the structural mapping, and it closely follows the specification presented in Section 3.

5 CONCLUSIONS

Multi-level modeling has a well-established literature in the modeling research community. In our particular branch of research, level-adjuvant approaches show strict level cohesion and segregation principles (Kühne, 2018), and they mostly rely on the potency notion as a concrete means of achieving deep instantiation. Due to its relative importance in this regard, the potency notion had been developed over the years, with a few variations existing in both theory and practice today. Hence, in this paper, we discussed a direct mapping between the main concepts of the classic potency notion and its DMLA representation, which is our level-blind multi-layer modeling framework. However, this paper is only the beginning of a series of research studies which we intend to publish where our aim is to provide a practical modeling playground with automatic model validation in order to be able

to better investigate and verify the semantics of many multi-level modeling concepts. Therefore, in our current work, we have implemented the concepts of the potency notion according to its original definition and presented how the mapping thereof was encoded in DMLA. The results are promising for a multi-level modeling framework to support future research work such as i) implementing more varieties and interpretations of existing multi-level concepts, and ii) introducing domain-specific languages to adopt level-adjuvant concepts in DMLA.

ACKNOWLEDGEMENTS

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications.

REFERENCES

Atkinson, C. and Gerbig, R. (2016). Flexible deep modeling with melanee. In *Modellierung 2016 - Workshopband : Tagung vom 02. März - 04. März 2016 Karlsruhe, MOD 2016*, volume 255, pages 117–121, Bonn. Köllen.

Atkinson, C., Gerbig, R., and Kennel, B. (2012). Symbiotic general-purpose and domain-specific languages. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1269–1272, Piscataway, NJ, USA. IEEE Press.

Atkinson, C., Gerbig, R., and Kühne, T. (2014). Comparing multi-level modeling approaches. In *CEUR Workshop Proceedings*, volume 1286.

Atkinson, C. and Kühne, T. (2000). Meta-level independent modelling. In *International Workshop on Model Engineering at 14th European Conference on Object-Oriented Programming*.

Atkinson, C. and Kühne, T. (2001). The essence of multilevel metamodeling. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 19–33, Berlin, Heidelberg. Springer-Verlag.

Atkinson, C. and Kühne, T. (2008). Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359.

Börger, E. and Stärk, R. (2003). *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag New York, Inc., 1st edition.

Clark, T. and Willans, J. (2012). Software language engineering with xmf and xmodeler. In *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, volume 2, pages 311–340.

de Lara, J. and Guerra, E. (2010). Deep meta-modelling with metadepth. In Vitek, J., editor, *Objects, Models, Components, Patterns*, pages 1–20, Berlin, Heidelberg. Springer Berlin Heidelberg.

de Lara, J., Guerra, E., and Cuadrado, J. S. (2014). When and how to use multilevel modelling. *ACM Transactions on Software Engineering and Methodology*, 24(2):12:1–12:46.

Fowler, M. and Scott, K. (2000). *UML Distilled (2Nd Ed.): A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Frank, U. (2014). Multilevel modeling - toward a new paradigm of conceptual modeling and information systems design. *Business & Information Systems Engineering*, 6(6):319–337.

Gerbig, R., Atkinson, C., de Lara, J., and Guerra, E. (2016). A feature-based comparison of melanee and metadepth. In *Proc. of the 3rd International Workshop on Multi-Level Modelling co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2016), Saint-Malo, France, October 4, 2016.*, pages 25–34.

Kühne*, T. (2018). Exploring potency. In *Proc. of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, pages 2–12, New York, NY, USA. ACM.

Kühne, T. (2018). A story of levels. In *Proceedings of MODELS 2018 Workshops, Copenhagen, Denmark, 2018*, pages 673–682.

Kühne, T. and Schreiber, D. (2007). Can programming be liberated from the two-level style: Multi-level programming with deepjava. *SIGPLAN Not.*, 42(10):229–244.

MDA (2001). OMG: Model Driven Architecture. https://www.omg.org/mda. Accessed:2019-10-23.

Mezei, G., Theisz, Z., Urbán, D., Bácsi, S., Somogyi, F. A., and Palatinszky, D. (2019). A bootstrap for self-describing, self-validating multi-layer metamodeling. In Dunaev, D. and Vajk, I., editors, *Proceedings of the Automation and Applied Computer Science Workshop 2019 : AACS'19*, pages 28–38.

MOF (2005). OMG: MetaObject Facility. http://www.omg.org/mof/. Accessed:2019-10-23.

Neumayr, B., Schuetz, C. G., Jeusfeld, M. A., and Schrefl, M. (2018). Dual deep modeling: multi-level modeling with dual potencies and its formalization in f-logic. *Software & Systems Modeling*, 17(1):233–268.

Urbán, D., Mezei, G., and Theisz, Z. (2017). Formalism for static aspects of dynamic metamodeling. *Periodica Polytechnica Electrical Engineering and Computer Science*, 61(1):34–47.

Urbán, D., Theisz, Z., and Mezei, G. (2018). Self-describing operations for multi-level meta-modeling. In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, pages 519–527. INSTICC, SciTePress.