Exploring Potency

Thomas Kühne
Victoria University of Wellington
P.O. Box 600, Wellington 6140, New Zealand
Thomas.Kuehne@ecs.vuw.ac.nz

ABSTRACT

The original notion of potency – one of the core features underpinning many forms of multi-level modeling – has come under pressure in several ways: First, since its inception new modeling challenges have come to the fore that raise serious questions about classic potency. Second, classic potency was developed in the context of constructive modeling and does not accommodate exploratory modeling, thus representing a major hindrance to the unification of constructive and exploratory modeling in a multi-level modeling context. Third, as the discipline of multi-level modeling has evolved, a number of alternative interpretations of potency have emerged. In part, these are based on different underlying principles, yet an explicit recognition of the respective differences at a foundational level and an explicit discussion of the trade-offs involved has been missing from the literature to date. In this paper, I identify limitations of classic potency, propose to evolve it to a potency notion based on a new foundation which – along with further novel proposals – addresses the aforementioned challenges, and finally conduct a comparison to three alternative definitions of potency.

CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**; • **Software and its engineering** → *Design languages*;

KEYWORDS

multi-level modeling, potency, level, order, constructive modeling, exploratory modeling

ACM Reference Format:
Thomas Kühne. 2018. Exploring Potency. In *Proceedings of ACM/IEEE 21th MODELS '18, October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3239372.3239411

1 INTRODUCTION

Multi-level modeling has been receiving a growing level of interest as a way of providing both flexibility and type-safety in modeling [7, 18]. The MODELS conference has already hosted five multi-level modeling workshops within the "MULTI" workshop series, and the recent Dagstuhl seminar 17492 was entirely devoted to multi-level modeling [1].

MODELS '18, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-4949-9/18/10…$15.00
https://doi.org/10.1145/3239372.3239411

Deep modeling, one of the earliest approaches to multi-level modeling, has inspired numerous related approaches, leading to

- employment in specific domains [15, 31],
- usage in industry [2, 17], and
- usage in standard definition initiatives [19].

Deep modeling can be thought of as being based on two major features: The alignment of metalevels with domain levels (as opposed to an alignment with language definition layers) and deep instantiation [6]. Both features have received widespread adoption among alternative multi-level modeling approaches [18, 25]. However, the particular way in which deep instantiation (i.e., the ability of an element to have an instantiation depth higher than one) is realized differs considerably among various approaches to multi-level modeling. In particular, the notion of *potency* is interpreted in fundamentally different ways.

In general, the goal of potency is to provide a basis for organizing model content through classification by providing the ability for model elements to characterize the properties of their instances over multiple levels of classification. In this paper I will focus specifically on *clabject potency* which pertains to a modeling entity's extent of control over subjacent modeling levels. Other kinds of *potency* which I will not consider here include *field potency* which pertains to slots/attributes/operations (cf. *durability* [3] and *intrinsic attributes* [15]), and *value potency* which pertains to the changeability of attribute values over classification levels (cf. *mutability* [3]).

The original notion of clabject potency ([4]) has been an essentially unmodified part of deep modeling to date and I will refer to it in short as *classic potency*. I will demonstrate a number of unresolved questions associated with clabject potency by confronting it with select modeling scenarios that have emerged from initiatives to inject an ontological foundation into multi-level modeling [11, 12] (see Sect. 3).

Another source of pressure on classic potency is the continued convergence of constructive multi-level modeling and exploratory modeling [9–11, 20, 21, 23, 26]. Classic potency is not applicable in exploratory modeling contexts and if at least some of its ideas are going to be relevant in future unified theories of multi-level modeling then there is a need to identify an alternative notion of potency that is not only tenable in both areas but provides value to them (see Sect. 4).

Finally, the emergence of alternative potency-based modeling approaches over time has led to variants of potency notions that rest on different foundations. Examples include "leap potency" [13], "dual deep modeling" [27], "cross-level relations" [9] and "non-potency elements" [17]. These alternatives not only implicitly suggest that classic potency is not fit to address the concerns the alternatives have been designed to address, but the rich plurality of available alternatives potentially poses a wider problem of creating

diversity that may hinder a cohesive growth of the discipline. Furthermore, multi-level modeling users and researchers are currently missing a clarification of the fundamental differences between the various potency dialects.

Overall, the new ideas presented in this paper – based on novel insights of how to

- understand differences between various potency definitions,
- define potency so that it meets recent challenges, and
- complement potency with additional modeling features.

– are intended to not only constitute a new potency variant fit to support the development of new unifying approaches to constructive and exploratory modeling, but, more importantly, inform future discussions on comparing and designing potency-based approaches.

2 BACKGROUND

The properties relevant to the subsequent discussions are a modeling element's *order*, *level*, and *classic potency*.

2.1 Order

The order of a modeling element can be regarded as a measure of its set-theoretic classification power, since it corresponds to the maximum depth of the "type-of" relationships originating from an element. Comparable to the "height"-property of a tree, the order of an element is therefore a measure of the maximum number of classification relationships between the element and its most remote instances, in all possible worlds. Pure instances, typically representing domain particulars, have order zero, since they do not classify any instances. Regular types in two-level technology have order one, as their instances do not classify any elements, etc.

Figure 1 depicts various elements along with their order values. Note that Product has order one since even though it does not directly classify any instances, it indirectly classifies MobyDick (through Book).

2.2 Level

The Dagstuhl seminar 17492 working group "Formal Foundations and Ontology Integration", comprising nine international researchers with a diverse representation of fields – observed that various definitions of "level" may be used in a multi-level modeling context [1]. No ruling on any preferred definitions was made but it was explicitly recognized that organizing elements into levels always follows one of either two schemes:

LS_1: element.order = element.level, or less strictly
LS_2: element.order ≤ element.level.

Figure 1 shows three levels, separated by two horizontal dashed level boundaries. Note that all four elements on the left hand side of Fig. 1 comply to both schemes. In contrast, the two elements on the right hand side only comply to LS_2.

In both schemes, the element level difference (Δlevel, i.e., the relative level distance as measured by the difference in level values between elements in the same classification branch) between related elements is identical but scheme LS_2 enables elements to be optionally shifted up the level hierarchy. I discuss some of the ramifications of this distinction in Sect. 5.

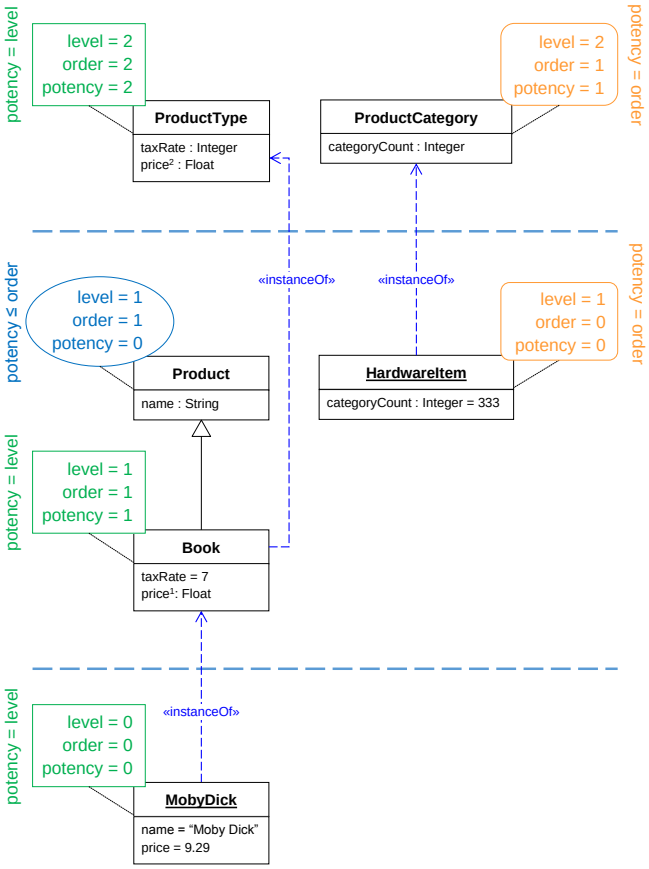

Figure 1: Level vs Order vs Potency

2.3 Classic Potency

Classic clabject potency controls the maximum number of instantiation steps available for an element, i.e., its maximum instantiation depth. This is achieved by codifying clabject potency as a non-negative integer value and enforcing the rule that any instance of a model element must have a potency that is one lower than that of its type [4]. Although this description bears a striking resemblance to the definition of *order* (cf. Sect. 2.1), there is a notable difference: While it is indeed rather common for an element to have element.potency = element.order (cf. Fig. 1 in which this applies to all elements but Product), in general, only the following holds:

P_1: 0 ≤ element.potency ≤ element.order.

Element *order* obviously cannot be lower than classic potency but classic potency semantics permits elements to have a potency that is lower than their order. Two distinct kinds of model elements in traditional object-oriented modeling are thus naturally associated with a potency of zero:

Abstract classes (e.g., Product^0) Only concrete classes may have direct instances whereas abstract classes are restricted to having indirect instances (through their subclasses).

Objects (e.g. MobyDick^0) Such elements do not have any instances in any shape or form.

3 ISSUES WITH CLASSIC POTENCY

In this section I present three modeling scenarios that highlight a number of unresolved questions surrounding classic potency.

3.1 Intermediate Abstract Class Inconsistency

Figure 2 shows a typical modeling scenario in which elements are arranged in a powertype configuration [30]. Breed plays the role of a powertype with respect to Dog, since all of Dog's subclasses are instances of Breed. The "{complete}" annotation near Corgi indicates that the *generalization set* formed by the subclasses of Corgi is "covering' ([29]), i.e., the superset implied by Corgi is completely partitioned by the subsets implied by PembrokeWelshCorgi and CardiganWelshCorgi. The corresponding modeling intention is to claim that any corgi individual must either be a pembroke welsh corgi or a cardigan welsh corgi. The superclass Corgi is only regarded as a useful generalization to reference corgis in general (all of which actually have a more specific type) whenever a distinction between the two corgi varieties is not required.

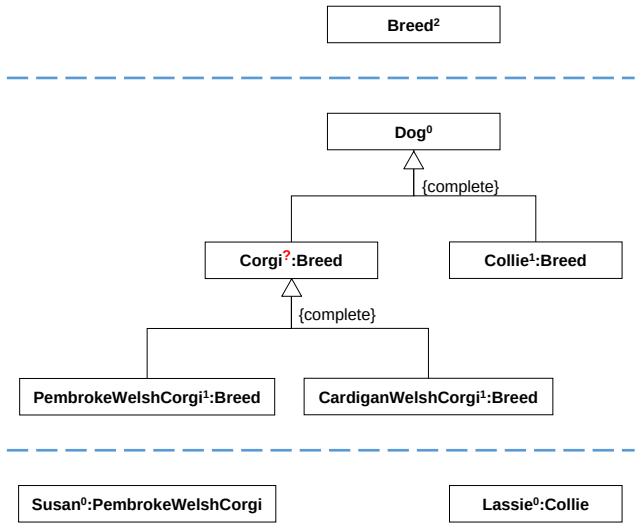

Figure 2: Intermediate Abstract Class Inconsistency

In an analogous case the UML superstructure specification 2.1.1 observes that Corgi, given the "{complete}" annotation, is an abstract class [28, p. 78]. This, in turn, suggests that Corgi should have potency zero. However, according to classic potency rules, Corgi should receive a potency of one (one less than Breed) because it is declared to be an instance of Breed. Due to these conflicting forces on the classic potency value, I marked the potency of Corgi as a question mark. At best, classic potency restricts modeler choice and, at worst, creates contradictory scenarios when interacting with covering generalization sets.

It may seem straightforward to avoid the conflict described above simply by not declaring Corgi to be an instance of Breed. This approach seems justifiable in that arguably Corgi has been relegated to a pure abstract notion and only its subclasses represent actual dog breeds. On the other hand, it would be useful for Corgi to receive all (potentially deep) features defined by Breed. Therefore, this attempt at a resolution shall not be considered to be a fully satisfactory solution in general.

3.2 Potency Zero Ambiguity

Figure 3 shows a model that demonstrates an ambiguity inherent to potency-zero elements according to classic potency. Deep modeling forgoes the notion of an explicit {abstract} tag for classes, as potency-zero classes essentially are abstract classes. As mentioned before, both objects and abstract classes have zero potencies, ergo it is not possible to conclude from a potency-zero value whether the element concerned is an object or an abstract class.

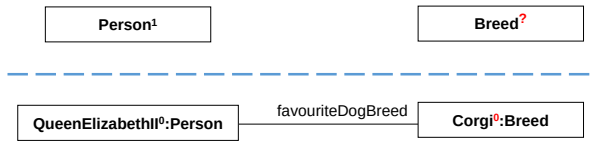

Figure 3: Potency Zero Ambiguity

In Fig. 3, Corgi's potency zero value could potentially denote that Corgi is an object, i.e., without further context, it is possible that Corgi : Breed is meant to be one of the many favorite things of the queen (e.g., among Blue : Color and Twinings: TeaSupplier) without any potential to ever serve a classifier role. Yet, its potency zero value does not preclude the possibility that it could be an abstract class that has (possibly elided) concrete subclasses, with the latter supplying instances that Corgi (indirectly) classifies (cf. Fig. 2).

Additionally, similar to the previous challenge, there is a potential conflict between the potency zero value of Corgi and the potency value of Breed. It is likely that the preferred potency value for Breed is two (cf. Fig. 2). However, this would require Corgi's potency to be one. I therefore put a question mark in place of Breed's potency.

3.3 Implicit Subclass Anomaly

So far, I have only considered complete generalization sets, i.e., subclasses were always meant to comprehensively cover all possible object types with the implication that no object may have the superclass as its direct type.

In Fig. 4(a), I now consider a model that contains a partitioning of dogs into subtypes that is declared to be incomplete. The respective "{incomplete}" tag should not be confused with model elision in which one presents a partial view on a complete model. The intent of "{incomplete}" is rather to express the possibility of dogs existing that are neither collies nor poodles [29, p. 123].

Figure 4(a) thus implicitly defines a third partitioning set – shown as a shaded area in the set visualization on the right hand side of Fig. 4(a) – containing all non-collie-non-poodle dogs. Arguably, the openness of the generalization set implies a *"catch-all" type* that provides a home to all dogs that are neither collies nor poodles. Figure 4(b) explicitly shows such a type and attaches the name "Mongrel" to it (disregarding any other pure dog breeds).

As it has been argued before, a class with a single "{complete}" generalization set – here Dog in Fig. 4(b) – should be declared as an abstract class, and hence should have potency zero. Yet, Dog in Fig. 4(a) cannot have potency zero since it is the only type available to account for non-collie-non-poodle dogs. Again, there are conflicting forces on the potency of Dog in Fig. 4(b) (cf. Sect. 3.1), given that a consistent treatment of Dog's potency in Fig. 4 seems to be a reasonable expectation.

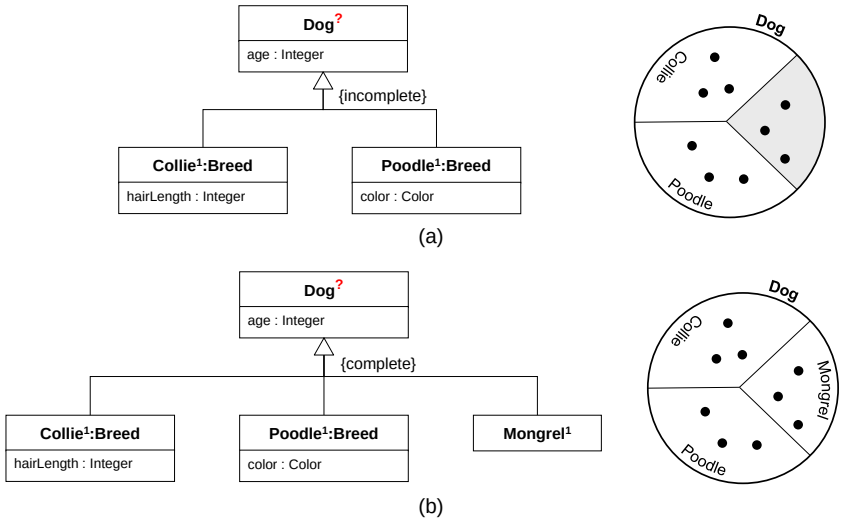

Figure 4: Implicit Subclass Anomaly

With respect to Fig. 4(b) suggesting a potency of zero, there is apparently an undesirable language feature interaction between classic potency and generalization set completeness. Such interactions should be avoided in general as they suggest a potential redundancy of language features and may lead to increased maintenance effort; singular changes may cause ripple-on changes due to the need to maintain consistency between interacting features.

More fundamentally, however, from an ontological perspective it is undesirable to treat Dog's potency as a volatile feature. Instead of Dog having a stable interpretation that encompasses all its features, its potency apparently depends on the context as shown in Figures 4(a) versus 4(b). This seems particularly undesirable as an incomplete partitioning (Fig. 4(a)) might be regarded as being equivalent to a complete one where an implicit *"catch-all" type* simply is not shown. Taking this view, Figures 4(a) & 4(b) are essentially depicting equivalent modeling scenarios, yet require different potency values for Dog. In the absence of fully satisfactory choices for both Dog occurrences in Fig. 4, I marked their potency values with question marks.

3.4 Summary

Arguably, in the light of the discussion so far, any clabject potency definition should fulfill at least the following requirements:

- R_1 support deep instantiation.
- R_2 be applicable in constructive and exploratory modeling modes.
- R_3 provide adequate expressiveness, i.e., adequately support common modeling scenarios (cf. Sect. 3.1–3.3).
- R_4 sustain a concise language design.

I maintain that any potency definition that aims to be relevant in the ongoing unification of constructive and exploratory modeling should at least satisfy requirements R_1-R_4.

None of the existing potency variants support all of R_1-R_4 and therefore one of the main contribution of this paper is to present a new definition of potency that satisfies R_1-R_4.

4 CHARACTERIZATION-POTENCY

At first sight an analysis of the issues with classic potency presented in Sect. 3.1-3.3 suggests that classic potency conflates the notions of *order* and *abstractness*. In traditional object-oriented modeling there is no conflict in making an order-one class abstract. Classic potency on its own, on the other hand, does not appear to support an independent combination of *order* and *abstractness*. For instance, in Fig. 2, by virtue of being an instance of Breed², Dog should be an order-one/potency-one class. However, since the *abstractness* of Dog cannot be simply asserted using an additional tag, but instead is expressed using potency zero, a conflict is introduced.

Interestingly, all other challenges dissolve as well when replacing classic potency with *order*, sometimes in combination with abstractness. The logical conclusion could therefore be to simply abandon the notion of potency, as distinct from *order*, since its only difference to *order* (cf. Sect. 2.3) introduces seemingly avoidable modeling conundrums. It turns out, though, that a new foundation for potency addresses the challenges and is more expressive than order-based schemes (see Sect. 5).

4.1 Characterization versus Classification

The crucial insight to understanding the issues with classic potency is to establish a differentiation between *order* and *potency* despite the considerable apparent similarities. Both notions relate to the length of "instance-of" chains in classification trees and an initial analysis only reveals that an element's potency is allowed to be smaller than its order (cf. Sect. 2.3). From that, it would be possible to conclude that the use of potency merely affords restricting an element's descriptive power to be less than its natural order would otherwise permit.

However, a much more profound difference can be established between *order* and *potency* that paves the path to a new definition of potency that not only resolves all previously mentioned issues but also is compatible with exploratory modeling. The key insight is to view *order* as pertaining to classification-depth and *potency* as pertaining to instantiation-depth.

In order to increase the clarity of the distinction between classification-depth and instantiation-depth, it is useful to change terminology and replace "instantiation" with "incarnation" to emphasize the creational aspect in contrast to passive classification. Likewise, an instance of a class shall be referred to as an *incarnation* of the class, if the instance was generated by the class (as would be the norm in a constructive modeling mode). In constructive modeling, usually the types come first and the instances are derived from them. In contrast, in exploratory modeling, the instances usually come first and the types are derived from them. As a result, the notion of *order* aligns well with exploratory modeling because abstractions like Book and Product naturally receive order-one status due to classifying an existing instance like MobyDick. In exploratory modeling, Product and Book are thus not qualitatively differentiated from each other as no distinction is made between a "direct classifier" versus an "indirect classifier". For instance, the nature of Book would not be changed by introducing ClassicBook as a subclass of Book and declaring MobyDick to be an instance of ClassicBook.

Note that in Fig. 1, Product – a class that would typically be declared as {abstract} – has classification-depth one (order = 1), as it indirectly classifies MobyDick. At the same time its instantiation-depth is zero, since – due to its abstract nature – it cannot have instances of its own (potency = 0).

Hence, the notion of *potency* aligns well with constructive modeling because in a constructive context classes are typically viewed as generating incarnations. In such a context, it is very natural to deliberate as to which types may be pure generalizations (here Product) and therefore are not designed to have any incarnations, versus other concrete types which are meant to have incarnations (here Book).

In OCL, the difference between incarnations (direct instances) and indirect instances manifests itself in the presence of two operators, oclIsTypeOf and oclIsKindOf that can be applied to an object [32]. While the former only evaluates to true if the object is a direct instance of the type, the latter also evaluates to true if the object is an indirect instance of the type.

In particular, with a view to accommodating exploratory modeling, however, it is useful to not focus on different types of instance-of relationships, but rather to recognize that the latter give rise to different class roles, i.e., to *characterizers* (that have incarnations) and *classifiers* (that have instances).

4.2 Accommodating Exploratory Modeling

Fortuitously, the distinction between characterizers and classifiers that was already instrumental in unifying constructive and exploratory typing disciplines [23], can also be the basis of a new definition of potency and thus achieve compatibility between potency and exploratory modeling as well as constructive modeling.

The class roles of *characterizer* and *classifier* imply two different kinds of "depth"-properties respectively: The characterization depth of a class in a characterizer role is the maximum depth of its *incarnation* relationships in all possible worlds. In contrast, the classification depth of a class in a classifier role is its maximum classification height with respect to all *instance-of* relationships associated with it in all possible worlds. Therefore, a useful differentiation between *order* and *potency* can be established by referring to the classification depth of a class as its *order*, and the characterization depth of a class as its *potency*.

Note that since *characterizer* and *classifer* are roles a class can play, rather than reflecting an intrinsic class property, it is possible for a single class to play both these roles simultaneously and thus meaningfully assume different values for its order and its potency respectively. Further, note that the differentiation of *potency* from *order* implies different perspectives: While the order of an element is determined by following instance-of chains from the deepest instance *upwards* the element, the potency of an element is determined by looking *downwards* along its incarnation chain.

These different perspectives make sense with respect to abstract classes which "break" incarnation chains. Arriving at an abstract class, when following incarnation chains downwards, results in the termination of exploring this particular incarnation branch, since abstract classes do not have incarnations themselves. In contrast, when determining the order of an element, it makes sense to follow incarnation chains upwards from the bottom to the element in question thus bypassing all abstract classes. N.B., in general one would follow instance-of relationships upwards, but in a *level-respecting* hierarchy ([22]) the maximum classification depth of an element coincides with the maximum height of incarnation relationships when navigating upwards from instances. Exploiting this property conveniently removes abstract classes from the equation when determining classification depth. Determining the order of abstract classes is then simply achieved by equating their order with the order of their subclasses.

In the following I use the new potency foundation to inform the resolution of the modeling challenges presented in 3.1–3.3. The respective potency mechanism will henceforth be referred to as *characterization-potency*.

4.3 Addressing the Intermediate Abstract Class Inconsistency

The model in Fig. 2 entails two challenges:

(1) From an exploratory perspective, Corgi should be available as a classifier. If a potency zero value were interpreted as removing Corgi as a classifier, it would clash with the exploratory world view that Susan is an instance of Corgi in the very same way it is an instance of PembrokeWelshCorgi.
(2) Corgi's potency, which should be "one" with respect to Corgi being an incarnation of Breed, should be "zero" with respect to Corgi's role as an abstract superclass.

The first of these two challenges, however, only arises when interpreting Corgi's potency as the order of a classifier. When using the previously established characterization depth semantics for Corgi's potency then there is no issue.

The first definition for characterization-potency is therefore

> *PD_1: The potency of a clabject is a measure of the latter's maximum incarnation depth in all possible worlds.*

Note the implied distinction between "potency" as a clabject property, and "potency constraints" as denoted through attaching superscript numbers to clabjects. The characterization-potency values encountered in respective diagrams actually represent constraints that establish an upper limit on an element's potency.

Also note that classic potency was intended as an "enabling" property, i.e., in order for a class to have a characterization depth higher than one, it was necessary to assign a potency value of two or more to it. Subsequently, Gerbig observed that this complicated a top-down approach for multi-level modeling as there may not be a-priori knowledge as to what the characterization depth for a class should be. In order to avoid repeated revisions to intermediate potency values, the notion of *star (∗) potency* was conceived to allow "wildcard" potency specifications [16].

I maintain that a "wildcard" essentially expresses non-committal, and that it is hence more elegant to instead draw on the notion of *underspecification* by interpreting the lack of any potency constraint as the absence of any restriction of an element's potency. This approach avoids

(1) clarifying the difference between "no potency value" and ∗-potency.
(2) a proliferation of elements with ∗-potencies, as these are not just found at the top level but may occur at any lower level due to the possibility of repeatedly deferring a commitment to a numeric potency by retaining "∗" as the potency value in an incarnation.
(3) an inadequate sense of arbitrariness regarding the potency value of the top element with a ∗-potency and an inadequate sense of all elements with a ∗-potency in an incarnation chain to posses the "same" potency.
(4) complicating potency semantics, since no special cases for handling a "∗" value need to be defined.

A downside to the use of underspecification is that the absence of any potency constraint can no longer represent a potentially convenient context-specific default value, e.g. the level value an element resides on. Given the fundamental importance of an element's potency, though, I argue it is beneficial to use respective constraints explicitly, rather than only recording deviations to a context-dependent default value.

For these reasons, elements should have unlimited potency (bounded by their order), in the absence of any explicit potency constraints:

> *PD_2: The potency of a clabject is equal to its order unless further restricted by a potency constraint.*

Addressing the second challenge listed above, requires fully embracing explicit potency values as constraints, rather than displaying (a potentially diminished) *order*. Classic potency was required to reduce by exactly one upon instantiation, as it was essentially treated as an instantiation depth counter. In contrast, characterization-potency only represents an upper bound on characterization depth. The third new potency definition is therefore:

> *PD_3: The potency of an incarnation is nonnegative and smaller than the potency of its characterizer.*

Overall, the new foundation for characterization-potency allows Breed's potency to be "two" in Fig. 2, despite the fact that Corgi's potency needs to be "zero" due to the generalization set constraint (cf. Sect. 3.1), meaning that all issues associated with the "Intermediate Abstract Class Inconsistency" challenge can be resolved by applying characterization-potency instead of classic potency.

4.4 Addressing the Potency Zero Ambiguity

Addressing the potency zero ambiguity discussed in Sect. 3.2 could be achieved by adding an additional language feature to indicate either *abstractness* or *objectness*. However, this would compromise R_4. Fortunately, a close examination of the suspected ambiguity reveals that there are other options.

Note that the potency-zero element Corgi^0 in Fig. 3 is declared to be an instance of Breed^2. Using characterization-potency instead of classic potency it becomes possible to replace the question mark in Fig. 3 with a "2" (cf. Fig. 2), as there is no longer an issue with instantiating a potency-zero Corgi from a potency-two metatype Breed. This apparent "jump" in potency values would only constitute a problem if characterization-potency were locked with *order*, as classic potency was. Even though classic potency was already allowed to deviate from *order* in terms of absolute values, it had to decrease exactly like *order* regarding relative decrements (Δpotency = Δorder). This was in accordance with sanity-enforcing constraints, such as *strict metamodeling* ([5]) or the *level-respecting* principle ([22]), that prevent an individual from being directly obtained from a metatype. As such, classic potency intentionally forbids directly incarnating an order-zero element Corgi^0 from an order-two metatype Breed^2. The new characterization-based foundation for characterization-potency further decouples potency from *order*, though, and thus supports Corgi^0 : Breed^2 and sanity-enforcing frameworks at the same time. The crucial insight is that tightening a characterization depth constraint more than strictly necessary (i.e., decreasing potency more than by one) is not in conflict with the order of the element being reduced by exactly one.

Note that since the *order* difference (Δorder) between two elements in an instance-of relationship must be exactly one, one can infer that Corgi^0 cannot be an object, if Breed's has potency two. An element with potency two (here Breed^2) must have an order of at least two (cf. P_1 in Sect. 2.3). This means that any of its incarnations (here Corgi^0) must have at least order one. In other words, if Breed is a potency-two element, there is no potency zero ambiguity, as the possibility for Corgi to be an object is ruled out.

A case could be made that a modeler should not be asked to perform such inferences, but this demur could be met by tools that support the (optional) display of an element's order. The latter can be mechanically computed from the incarnation tree in many cases (if there are level-zero "anchors" to start from) and in others either a lower bound could be shown or the multi-level modeling language in question could be extended to support specifying *objectness* in which case order could always be fully determined (see Sect. 5).

An ambiguity regarding Corgi's nature could still occur if Breed's potency is chosen to be "one". In the absence of any order information about Breed^1 this could technically allow entertaining the idea of Corgi^0 as an object. Even then, however, it is still possible to ascertain whether Corgi is an object or an abstract type, with the exception of one (rather pathological) case.

If Corgi^0 were to represent an abstract *type*, one would expect it to have a type facet, i.e., declare features, e.g., those that are common to all its subtypes (cf. Fig. 2), such as age^1(), coatColor^1(), etc. From the presence of such potency-one features one would immediately be able to determine that Corgi^0 cannot be an object but must be an abstract type instead. In other words, the potency zero ambiguity

may only arise if a potency-zero type has a vacuous type facet, i.e. when it has no features with a potency higher than zero. Yet, the absence of an effective type facet that imposes constraints on instances, would imply that the only classifier role the type could play would be a degenerate one. The utility of such a universal classifier that does not stipulate any kind of requirements on its instances is debatable and represents a weak case to support the introduction of an additional language feature to capture *objectness*.

Finally, regarding the above described pathological case, ambiguity should not be immediately regarded as a shortcoming. Ambiguity can be an advantage when it can be usefully interpreted as underspecification in order to avoid premature overcommitment. Just as deep modeling employs level-agnosticism in order to obtain a uniform notation and application of principles across all classification levels, one can regard rare occurrences of potency zero ambiguity as a useful form of avoiding overspecification. In Fig. 3 it does not really matter whether Corgi has order zero, i.e., is an object, or not. All that matters is that Corgi is referenced by the queen as one of her favorite things. There is no natural requirement that the queen must only prefer particulars as opposed to universals. Not even the *strict metamodeling* doctrine ([5]), which is sometimes viewed as "too restrictive", establishes a requirement that all elements in a level need to have the same order.

4.5 Addressing the Implicit Subclass Anomaly

In Sect. 3.3, I questioned whether conflicting forces on Dog's potency in Figures 4(a) & 4(b) imply that the notion of potency is not ontologically stable. The first insight towards addressing this challenge concerns the recognition of the actual property in question, which is Dog's *abstractness* rather than its potency. If there is no resolution of the conflicting forces of whether or not Dog should be an abstract class, a wider issue would be at stake in which case an issue with potency – note that the new definition still subsumes *abstractness* – would just constitute collateral damage.

The following analysis reveals that there is no need to modify the new definition of characterization-potency. Instead, the analysis ultimately suggests a potential symbiotic cohabitation between potency and generalization set completeness rather than the hitherto assumed mutually constraining feature interaction. First, however, I resolve the conflicting forces on Dog's *abstractness*.

In order to see why Figures 4(a) & 4(b) do not endanger the stability of Dog's *abstractness* (and hence potency value), it is necessary to understand that they do not show equivalent scenarios. Although Fig. 4(a) implies a complement-*set*, i.e., the set of all dogs that are neither collies nor poodles, it does not imply a respective (complement-) *type*. In particular in a nominal typing paradigm, the existence of a *set* does not imply that there is a *type* – anonymous or not – with said set as an extension.

The explicit introduction of type Mongrel in Fig. 4(a) therefore establishes a qualitative difference that explains why Dog's abstractness/potency has to change. Instances of relevance to the modeler must be associable with types contained in a model. Hence, in the absence of any other dog breeds in Fig. 4(a), Dog must have potency "one" in order to accommodate dog incarnations that could be mongrels or, as a matter of fact, instances of an as yet unmentioned pure breed, such as Labrador.

In contrast to Fig. 4(a), Fig. 4(b) specifies that all possible dog kinds have been enumerated (through the "{complete}" tag), implying that no dog should be accommodated as "just a dog", meaning that Dog's potency should be "zero". Note how cleanly the new definition of characterization-potency handles the fact that while Dog^0 is unavailable as a *characterizer* (i.e., for creating dog incarnations or accommodating instances that are "just dogs"), it may still be used as an *order-one classifier* (i.e., can be used to generically refer to all kinds of dog instances of a more specific type).

The above reasoning explains why an apparently "inconsistent" potency assignment to Dog in Figures 4(a) & 4(b) is not a sign of an ontologically unstable Dog conceptualization but rather accurately reflects different modeler intents. As a matter of fact, however, a further analysis suggests that a "{complete}" tag could be associated with a more useful interpretation in frameworks that only allow single classification, i.e., do not support instances with multiple direct types.

4.5.1 Completeness Coupling. If *completeness* of a generalization set is interpreted in the sense of the UML's "*covering*" property [29, Sect. 9.9.8.4, Tab. 9.1] then it establishes a strong coupling between the completeness of a generalization set and the *abstractness* of the superclass. It does not make sense for the superclass to be concrete if at least one of its associated generalization sets is marked as {complete}. Any instance of the superclass would be forced to have a more specific type, i.e., one that is at least as specific as one of the subclasses in the {complete} generalization set. If, on the other hand, all associated generalization sets are marked as {incomplete} then the superclass cannot be abstract because it would be impossible to accommodate instances that do not fit any of the subclasses. Technically, such accommodation could be provided if the abstract superclass had concrete superclasses. This would not make any sense from an ontological perspective, though, as one would characterize an instance with a generic type but deny it a more specific characterization despite the fact that it is recruited from the more specific sub-domain implied by the supertype and its associated subtypes. As a result, I propose a different interpretation of {complete} and {incomplete} tags which lifts the notion of an *open world assumption* from the level of instances to the level of types.

Discussions regarding the completeness of generalization sets are trivial when a closed world assumption can be made and the modeler is omniscient. In this case all generalization sets could be made complete without loss of generality. However, particularly in exploratory contexts with an *open world assumption*, there is no a priori knowledge about which instances exist in the universe of discourse and which types may be needed to properly accommodate them. Only in special cases will it be possible to make an a priori stipulation as to which generalization sets are complete. For instance, an exclusive partitioning of the concept Screw into LeftHandedScrew and RightHandedScrew is safe because there is absolute certainty that no other forms of screws will require accommodation.

I propose to use "{complete}" tags in such cases only. In other words, "completeness" would no longer be concerned with whether instances need to potentially be accommodated by the supertype or not. The latter aspect is addressed by making the supertype either abstract or concrete. This frees the "completeness" of a generalization set to make an ontological commitment as to whether

the specified subclasses are known to reliably exclude any other possible subtypes, in all possible worlds.

The four possible combinations of generalization set completeness and superclass abstractness are briefly discussed below.

4.5.2 Complete with Abstract/Concrete Supertype.
In Fig. 4(b), using the proposed revised understanding of "{complete}", the respective model just conveys that dog breeds not mentioned in the model are irrelevant for all stakeholders involved. This denies any dog of an unmentioned breed (such as Labrador) to ever be characterized more specifically than Dog. However, there is still a choice as to whether or not such dogs can be accommodated (i.e., be accepted as an incarnation of a type in the model) or not. If such dogs are considered to be inadmissible as Dogs then one would choose Dog^0 (cf. Fig. 4(b)); if they are admissible, one would choose Dog^1. Only the former case could hitherto be expressed using the traditional interpretation based on "covering" subtypes. The latter, new case represents the choice to make an ontological commitment that the listed subtypes are the only ones that will ever matter for any specific characterization while still allowing the accommodation of incarnations that are "just dogs".

Particularly in exploratory contexts, the above described control supports an a priori determination of whether found instances are going to be of relevance to the modeler or not, and if so, at what level of specificity.

4.5.3 Incomplete with Abstract/Concrete Supertype.
The case of an incomplete generalization set and a concrete supertype corresponds to Fig. 4(a). It is very similar to the case of having a complete generalization set and a concrete supertype but differs from the latter by not rejecting any other subtypes from ever making sense. Thus one would choose "{incomplete}" rather than "{complete}" whenever one wants to communicate that a future extension to already mentioned subtypes is possible, if not expected.

The remaining case, comprising an incomplete generalization set and an abstract supertype, cannot be expressed with UML's "covering" semantics either. A motivating example is a context (e.g., an animal shelter) in which it is necessary to have specific knowledge about a dog, i.e., what its actual breed is, because treating it generically as a Dog would be inadequate. In this case, assigning potency zero to Dog in Fig. 4(a), makes Dog unavailable as a characterizer, thus ruling out inadequate treatment, with the understanding that only fully characterized dogs can be treated. Yet, such a choice now does not imply the (false) ontological overcommitment of claiming that the specified subtypes are "{complete}", i.e., that all dogs in all worlds may only have one of the mentioned breeds.

4.6 Summary

The analysis of the "implicit subclass anomaly" resulted in the recognition of the fact that variable potency choices – affecting the abstractness of a superclass – are not a sign of ontological instability but a reflection of model semantics. This gives modelers the highly desirable ability to explicitly communicate their intent. Furthermore, assuming a single classification framework, it became clear that there is an unnecessary coupling between superclass abstractness and generalization set completeness which could be addressed by letting "completeness" refer to the exclusivity of subtypes, rather than the coverage of subtypes. In a multiple classification framework and the presence of multiple generalization sets associated to one superclass, it would become necessary to restore some way to mandate that an incarnation has to be recruited from a particular generalization set. The "covering" approach accomplishes that, but it appears that a separate notion, e.g., using a "{one-of}" tag, would be preferable. The advantage would be that "{one-of}" could be combined with either "{complete} or "{incomplete}, thus allowing to force recruitment from a particular branch while giving the modelers the option to specify whether or not the subtypes in that branch are exclusive or open to future extension.

Overall, independently of the finer choices available for ontological commitments regarding generalization sets, it has become clear that the new foundation chosen for characterization-potency, derived from ideas originally developed for unifying constructive and exploratory typing disciplines, not only makes characterization-potency fit for use in unified multi-level frameworks supporting both constructive and exploratory modeling modes, it furthermore addresses all issues that were previously identified regarding classic potency (see Sect. 3).

5 COMPARISON

Analyzing the literature of potency-based multi-level modeling approaches reveals that there are four different categories of potency interpretations:

level-potency: potency = level, e.g., [27].
order-potency: potency = order, e.g., [14, 25].
order-locked-potency: Δpotency = Δorder, e.g., [4].
decoupled-potency: potency $\leq$ order (cf. Sect. 4).

All of them support R_1 (cf. Sect. 3.4), therefore the following discussion will focus on R_2-R_4.

5.1 Level-Potency

If an approach chooses to let *potency* coincide with *level*, it follows (from P_1 (cf. Sect. 2.3) and either LS_1 or LS_2 (cf. Sect. 2.2)) that potency = order = level. All left-hand-side elements in Fig. 1, with the exception of Product, comply to this scheme.

The notion of level-potency is distinguished from all other approaches in that it does not allow the two classes on the right hand side of Fig. 1 to reside that high in the level hierarchy. This pair would have to move down one level to comply with a level-potency approach. Although HardwareItem appears to be a regular object to all intents and purposes and would thus appear to be better placed at the bottom level, not allowing the placement in Fig. 1 is undesirable for a number of reasons.

First, HardwareItem has a different ontological foundation than ordinary objects, such as MobyDick. The categoryCount value captured by HardwareItem (333) represents an abstract count (an aggregate over all hardware sales). An analogous count value at the level of an object residing at the level below would represent a concrete count, such as the number of sales of a book. Second, HardwareItem could entertain an association with Product or another class at Product's level. In a level-potency scheme, this would violate the *strict metamodeling* doctrine, or other similar sanity-enforcing principles, because HardwareItem could not reside at the same level as its

associated classes. Third, it would be a very reasonable design to make ProductType a subclass of ProductCategory, and HardwareItem a subclass of Product (cf. [24, Fig. 8]). Again, well-formedness schemes intended to provide a means for sanity-checking models (such as *strict metamodeling*), would only permit these subclass relationships if they are intra-level relationships. Note that Brasileiro et al. found that over 87% of the classes in Wikidata multi-level taxonomies were involved in ill-formed configurations that could have been picked up by employing sanity checking [8], giving credence to the utility of establishing and enforcing well-formedness rules for classification hierarchies.

5.2 Order-Potency

If an approach chooses to let *potency* coincide with *order*, it follows – from P_1 (cf. Sect. 2.3) and LS_2 (cf. Sect. 2.2) – that potency = order, and order ≤ level (choosing LS_1 instead of LS_2 would result in the more constraining level-potency approach). By decoupling potency and order from level, order-potency is therefore more flexible and does not entail the aforementioned disadvantages of level-potency. Furthermore, both order-potency and level-potency – unlike classic potency – satisfy R_2, i.e., they both cater to exploratory modeling as their foundation on *order* aligns well with the *classifier* semantics assumed in exploratory modeling. On the downside, however, they do not cater to constructive modeling, as they do not support any notion of *characterization*, i.e., the distinction between the incarnations of a class and its (indirect) instances. Only characterization-potency provides exploratory as well as constructive modelers with a means to differentiate between different type roles – from pure abstractions, over to general descriptions, to sufficiently characterizing types (see Sect. 4.1) – with all associated benefits [23].

Note that both order-potency and level-potency would force order-one class Product in Fig. 1 to have potency one and it would therefore no longer be possible to encode abstractness of types with a potency zero value. As a result, to enable abstractness of Product, a different mechanism (e.g. employing an "{abstract}" tag) is required. This impacts negatively on R_4, compared to characterization-potency.

5.2.1 Ease of Use. Arguably though, language minimality as represented by R_4 is only a proxy for the ultimate goal of a language/framework that modelers can use competently and with ease. Solely considering the number of language constructs could be too simplistic as it may also matter how easily language constructs can be adequately applied.

Indeed, both order-potency and level-potency – due to offering fewer degrees of freedom – would have made the potency value discussions in Sect. 3 superfluous and thus may suggest superior ease of use. All potency values would have been mechanically derivable from the context, thus allowing the modeler to focus on other activities. In this sense, level-potency is the easiest to apply since it simplifies the potential handling of three variables – level, order, and potency – to the handling of a single variable.

However, as discussed in Sect. 5.1, such a simplification comes at the price of losing the option to enforce sanity-checking through well-formedness rules and, in general, expressiveness. While the much less restrictive characterization-potency approach requires modelers to make more choices, the outcome is also better ontologically-motivated (see Sect. 4.5 & 5.1). Ease of use (cf. R_4) therefore needs to be balanced with expressiveness (cf. R_3).

5.2.2 Expressiveness. Characterization-potency is the most expressive variant since decoupling potency from order allows to control the *characterization* versus the *classification* aspects of a type independently from each other (cf. Sect. 4.1). Furthermore, it does not require an additional *abstractness* concept which would impact on R_4. While in all examples shown in this paper it is possible to achieve the effect of a "characterization-potency = 0" scenario by using an {abstract} tag with level- or order-potency, a modeler has to manually assign such tags. In contrast, characterization-potency supports the *automatic*, i.e., predetermined attribution of *abstractness* once a certain incarnation depth is reached. This affords support for "deep characterization" as it is possible to essentially set a "timer" on an element that controls when – in terms of incarnation depth – it is no longer considered to be suitable as a characterizer. The latter property will typically correlate with the depth of the element's type facet, i.e., how high its feature-potency values are. Characterization-potency thus affords modelers control from higher levels over lower levels that neither order-potency nor level-potency provide.

Characterization-potency on its own, however, is less expressive in terms of constraining an element to be purely an object. In general, the following modeling needs may apply:

a) there is never a pragmatic need to distinguish between objects and abstract classes.
b) it is satisfactory to infer the absence of *objectness* from the presence of a non-vacuous type facet and/or the type's potency (cf. Sect. 4.4).
c) it is sometimes necessary to narrow down the role of an element to that of a pure object.

If the first alternative applies, it is – in the sense of underspecification (cf. Sect. 4.4) – desirable that the model in Fig. 3 can simply express the fact that Corgi cannot have incarnations without espousing this fact by invoking (and thus committing to) either *objectness* or *abstractness*. It may well be the case that Corgi is initially only conceived as an object, until it is discovered that it is ideally suited to be a classifier for instances like Susan as well. A potency-based characterization of Corgi then requires no modification whereas more committal specifications would have to be revised.

If the second alternative from the above list applies then characterization-potency on its own would be sufficient. If the third alternative applies then characterization-potency would have to be enhanced with an *objectness* construct.

A straightforward *objectness* design would employ a simple {object} tag, however, a more general approach would be to have the option of constraining *order*, i.e., classification depth. An "order = 0" constraint would fully constrain the role of an element to that of an object (in contrast to "potency = 0" which would allow the element to have subclasses). An alternative is to constrain *cardinality*, i.e., the maximum number of instances an element may have. An element with "cardinality = 0" is an object, an element with "cardinality = 1" is a singleton type. Higher values and cardinality ranges could be useful in the context of embedded systems which often imply limitations on the number of elements of a certain type [33]. With

any of the aforementioned additions, it would be possible to use characterization-potency constraints for underspecification when useful and a tighter specification when it is absolutely necessary to exclude (super-)type roles for modeling elements.

5.3 Order-Locked-Potency

Classic potency uses a notion of potency that is distinct, but not entirely decoupled, from *order*. It uses P_1 and LS_2, therefore resulting in potency ≤ order ≤ level. Classic potency instantiation rules, however, establish a further constraint: Δpotency = Δorder, i.e., potency must decrease along with order in a lock-step fashion. So while classic potency offers more flexibility than either level-potency or order-potency, it does not satisfy R_3 (cf. Sect. 3).

5.4 Summary

It turns out that all four potency categories considered in this section can be ordered according to the degrees of freedom they provide to modelers. Order-potency strictly subsumes level-potency, i.e., it supports the same scenarios plus additional ones, thus adding expressiveness. Order-locked potency adds further modeling scenarios that can be supported but is not strictly adding to expressiveness. While the idea of *potency* as being distinct from *order* opens up further scenarios, it also implies some limitations if one chooses to forgo an orthogonal notion of *abstractness* (cf. Sect. 3).

By virtue of being based on a new foundation, characterization-potency provides the highest level of flexibility without suffering the limitations of any of the other alternatives.

6 CONCLUSION

The original potency notion for multi-level modeling ([4]) has not received any radical updates to its definition since its inception over almost two decades ago and still performs well in the constructive modeling context it was conceived for. However, over time, a number of modeling challenges have been identified (see Sect. 3.1-3.3) that raised questions as to whether the very idea of potency was tenable, in particular in exploratory modeling scenarios.

Characterization-potency, as proposed in this paper, can be regarded as an evolution of classic potency that rests on a new foundation, drawing on a distinction between *characterization* and *classification* which was originally developed for the purpose of unifying constructive and exploratory typing disciplines [23]. Transferring these concepts to multi-level modelling allowed the insight that potency should not be understood as merely allowing an element to relinquish (some of) its instantiation depth (cf. Sect. 2.3), but rather as a means to control an element's characterization depth. This seemingly subtle difference has a number of deep reaching consequences (cf. Sect. 4 & 5):

- characterization-potency cannot be misunderstood as confounding *order* with *abstractness*: An element may meaningfully have different values for *order* (its classification depth) and potency (its characterization depth).
- when Δpotency is not synchronized with Δorder, many issues with classic potency are resolved.
- potency values do not enable deep instantiation, but constrain characterization depth, and thus obviate ∗-potency.
- characterization-potency is uniquely expressive while maintaining a minimal language design.
- separating characterization depth from classification depth supports potency-based deep characterization in unified constructive and exploratory frameworks.

Due to the last aspect, characterization-potency allows both exploratory and constructive modelers to acknowledge and express different type roles; from pure abstractions, over general and concrete descriptions, to sufficiently characterizing types. Unlike any of the other potency variants, characterization-potency is not only compatible with exploratory modeling but enriches it with *deep* characterization control. Note that characterization is a meaningful idea even in the absence of any constructive notion of instantiation ([23]), which makes characterization-potency applicable even in purely exploratory approaches, such as ontologies.

Although the new foundation for characterization-potency and the associated novel ideas regarding generalization set completeness (cf. Sect. 4.5) and generalizations of *objectness* specification (cf. Sect. 5.2.2) are contributions in their own right, the comparison of characterization-potency to existing potency variants may be at least as significant for future multi-level research.

Many multi-level modeling languages/frameworks have taken classic potency ([4]) as an inspiration, leading to the proposal of a considerable number of potency variants. However, to date comparisons between these variants have only observed their usage with respect to select modeling scenarios. This paper presents the first systematic analysis of four different categories of potency schemes at a fundamental level (cf. Sect. 5), and thus, for the first time, explicates fundamental differences that hitherto were only implicitly manifested.

In my analysis I established that existing potency schemes can be ordered according to the degrees of freedom they provide to modelers in terms of decoupling potency from *order* and *level*. I then discussed the trade-offs involved in terms of language minimality, ease of use, and expressiveness.

Summarizing, the presented work should improve multi-level modeling approaches through a new notion of *characterization-potency*, further the unification of exploratory and constructive modeling by providing deep characterization, and provide a new basis for future comparisons and further developments. The novel insights obtained through both designing characterization-potency and investigating the relationships between *potency*, *order*, and *level* should guide future efforts in language design and will hopefully lead to a consolidation of potency notions rather than further proliferation. Such an effect could prove to be crucial for the future growth of multi-level modeling.

ACKNOWLEDGMENTS

I could not have possibly written this paper without the invaluable input by Colin Atkinson. I would like to thank him in particular for numerous insightful debates on the topics presented here and for communicating the modeling challenges in Sect. 3.2–3.3. I would furthermore like to thank Victorio A. Carvalho and João Paulo Almeida for the formulation of the modeling challenge described in Sect. 3.1 and the "Formal Foundations and Ontology Integration' group of Dagstuhl Seminar 17492 for the stimulating discussions.

REFERENCES

[1] João Paulo A. Almeida, Ulrich Frank, and Thomas Kühne. 2018. Multi-Level Modelling (Dagstuhl Seminar 17492). *Dagstuhl Reports* 7, 12 (2018), 18–49. https://doi.org/10.4230/DagRep.7.12.18

[2] Thomas Aschauer, Gerd Dauenhauer, and Wolfgang Pree. 2009. Representation and Traversal of Large Clabject Models. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems, MODELS 2009, Denver, CO, USA, October 4-9, 2009*., Andy Schürr and Bran Selic (Eds.). Springer Verlag, 17–31. https://doi.org/10.1007/978-3-642-04425-0_3

[3] Colin Atkinson, Ralph Gerbig, and Bastian Kennel. 2012. Symbiotic General-purpose and Domain-specific Languages. In *Proceedings of the 34th International Conference on Software Engineering* (2012-01-01) *(ICSE '12)*. IEEE Press, Zurich, Switzerland, 1269–1272.

[4] Colin Atkinson and Thomas Kühne. 2001. The Essence of Multilevel Metamodeling. In *Proceedings of the 4th International Conference on the UML 2000, Toronto, Canada (LNCS 2185)*, Martin Gogolla and Cris Kobryn (Eds.). Springer Verlag, 19–33. https://doi.org/10.1007/3-540-45441-1_3

[5] Colin Atkinson and Thomas Kühne. 2001. Processes and Products in a Multi-Level Metamodeling Architecture. *International Journal of Software Engineering and Knowledge Engineering* 11, 6 (2001), 761–783. https://doi.org/10.1142/S0218194001000724

[6] Colin Atkinson and Thomas Kühne. 2007. A Tour of Language Customization Concepts. In *Advances in Computers*, Marvin Zelkowitz (Ed.). Vol. 70. Academic Press, Elsevier, Chapter 3, 105–161. https://doi.org/10.1016/S0065-2458(06)70003-1

[7] Colin Atkinson, Thomas Kühne, and Juan de Lara. 2017. Theme Issue on Multi-Level Modeling. SoSyM Theme Issue. Springer Verlag.

[8] Freddy Brasileiro, João Paulo A. Almeida, Victorio A. Carvalho, and Giancarlo Guizzardi. 2016. Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata. In *Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion)*. International World Wide Web Conferences Steering Committee, 975–980. https://doi.org/10.1145/2872518.2891117

[9] Victorio A. Carvalho and João Paulo A. Almeida. 2016. Toward a well-founded theory for multi-level conceptual modeling. *Software & Systems Modeling* (2016), 1–27. https://doi.org/10.1007/s10270-016-0538-9

[10] Victorio A. Carvalho, João Paulo A. Almeida, Claudenir M. Fonseca, and Giancarlo Guizzardi. 2015. Extending the Foundations of Ontology-Based Conceptual Modeling with a Multi-level Theory. In *Proceedings of ER'15*. Springer Verlag, 119–133. https://doi.org/10.1007/978-3-319-25264-3_9

[11] Victorio A. Carvalho, João Paulo A. Almeida, Claudenir M. Fonseca, and G. Guizzardi. 2017. Multi-level ontology-based conceptual modeling. *Data & Knowledge Engineering* 109 (2017), 3–24.

[12] Victorio A. de Carvalho, João Paulo A. Almeida, and Giancarlo Guizzardi. 2014. Using Reference Domain Ontologies to Define the Real-World Semantics of Domain-Specific Languages. In *Advanced Information Systems Engineering*, Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff (Eds.). Springer Verlag, 488–502.

[13] Juan de Lara and Esther Guerra. 2010. Deep Meta-modelling with MetaDepth. In *Proceedings of TOOLS (48)*. 1–20.

[14] Claudenir M. Fonseca, João Paulo A. Almeida, Giancarlo Guizzardi, and Victorio A. Carvalho. 2018. Multi-Level Conceptual Modeling: From a Formal Theory to a Well-Founded Language. In *Proceedings of the 37th International Conference on Conceptual Modeling (ER 2018) (LNCS)*. Springer Verlag.

[15] Ulrich Frank. 2014. Multilevel Modeling - Toward a New Paradigm of Conceptual Modeling and Information Systems Toward. *Business & Information Systems Engineering* 6, 6 (2014), 319–337. https://doi.org/10.1007/s12599-014-0350-4

[16] Ralph Gerbig. 2011. *The Level-agnostic Modeling Language: Language Specification and Tool Implementation.* Master's thesis. University of Mannheim, Mannheim, Germany. https://ub-madoc.bib.uni-mannheim.de/37153

[17] Muzaffar Igamberdiev, Georg Grossmann, Matt Selway, and Markus Stumptner. 2016. An integrated multi-level modeling approach for industrial-scale data interoperability. *Software & Systems Modeling* (2016), 1–26. https://doi.org/10.1007/s10270-016-0520-6

[18] Muzaffar Igamberdiev, Georg Grossmann, and Markus Stumptner. 2016. A Feature-based Categorization of Multi-Level Modeling Approaches and Tools. In *Proceedings of the 3rd Workshop on Multi-Level Modelling co-located with the 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2016) (CEUR Workshop Proceedings)*, Vol. Vol-1722. 45–55.

[19] Andreas Jordan, Matt Selway, Georg Grossmann, Wolfgang Mayer, and Markus Stumptner. 2014. *Re-engineering the ISO 15926 Data Model: A Multi-level Metamodel Perspective.* Springer International Publishing, 248–255. https://doi.org/10.1007/978-3-319-06859-6_22

[20] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. 2006. Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In *Proceedings of MODELS'06* (2006-12-07) *(LNCS)*, Vol. 4199. 528–542.

[21] Bastian Kennel. 2012. *A Unified Framework for Multi-Level Modeling.* Ph.D. Dissertation. University of Mannheim.

[22] Thomas Kühne. 2006. Matters of (Meta-) Modeling. *Software and System Modeling* 5, 4 (2006), 369–385. https://doi.org/10.1007/s10270-006-0017-9

[23] Thomas Kühne. 2018. Unifying Nominal and Structual Typing. *Software & Systems Modeling* (Feb 2018). https://doi.org/10.1007/s10270-018-0660-y

[24] Thomas Kühne and Daniel Schreiber. 2007. Can programming be liberated from the two-level style? – Multi-level programming with DeepJava. In *Proceedings of the 22nd ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, R. P. Gabriel, D. F. Bacon, C. Videira Lopes, and G. L. Steele Jr. (Eds.). ACM, USA, 229–244. https://doi.org/10.1145/1297105.1297044

[25] Juan De Lara, Esther Guerra, and Jesús Sánchez Cuadrado. 2014. When and How to Use Multilevel Modelling. *ACM Transactions on Software Engineering and Methodology* 24, 2, Article 12 (2014), 46 pages. https://doi.org/10.1145/2685615

[26] Bernd Neumayr and Michael Schrefl. 2009. Multi-level Conceptual Modeling and OWL. In *Proceedings of the ER 2009 Workshops (CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS) on Advances in Conceptual Modeling - Challenging Perspectives (ER '09)*. Springer-Verlag, 189–199. https://doi.org/10.1007/978-3-642-04947-7_23

[27] Bernd Neumayr, Christoph G. Schuetz, Manfred A. Jeusfeld, and Michael Schrefl. 2016. Dual deep modeling: multi-level modeling with dual potencies and its formalization in F-Logic. *Software & Systems Modeling* (2016), 1–36. https://doi.org/10.1007/s10270-016-0519-z

[28] OMG 2007. *Unified Modeling Language Superstructure Specification, Version 2.1.1.* OMG. OMG document formal/07-02-05.

[29] OMG 2017. *Unified Modeling Language Specification, Version 2.5.1.* OMG. OMG document formal/17-12-05.

[30] Chris Partridge, Sergio de Cesare, Andrew Mitchell, and James Odell. 2018. Formalization of the classification pattern: survey of classification modeling in information systems engineering. *Software & Systems Modeling* 17, 1 (February 2018), 167–203. https://doi.org/10.1007/s10270-016-0521-5

[31] Christoph Georg Schütz, Bernd Neumayr, and Michael Schrefl. 2015. Multilevel Modeling for Business Process Automation. In *Proceedings of the 8th International Workshop on Evolutionary Business Processes (EVL-BP 2015) in conjunction with the 19th IEEE Conference on Enterprise Computing Conference (EDOC 2015), Adelaide, Australia.* IEEE Computer Society, 51–60.

[32] Jos Warmer and Anneke Kleppe. 1998. *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley.

[33] Manuel Wimmer, Petr Novak, Radek Sindelar, Luca Berardinelli, Tanja Mayerhofer, and Alexandra Mazak. 2017. Cardinality-Based Variability Modeling with AutomationML. In *Proceedings of ETFA 2017*. IEEE, 1–4.