Deep Specialization

Integrating Powertypes into Deep Modeling

Thomas Kühne* , Arne Lange†
* *Victoria University of Wellington*, Wellington, New Zealand
† *Karlsruhe Institute of Technology*, Karlsruhe, Germany

***Abstract*—Deep modeling, the potency-based approach to multi-level modeling, so far has not featured a *powertype* mechanism similar to those that can be found in alternative multi-level modeling approaches. While this does not represent a functional limitation, certain modeling scenarios favor the pragmatics of *powertype*-structures. We propose *deep specialization* as a means to achieve the effects of a powertype construct in deep modeling languages such as LML and DLM. Since deep specialization can be regarded as emerging from existing deep modeling concepts, we can avoid the introduction of a dedicated powertype construct. Beyond that, due to its generality, our mechanism even supports the specification of *subordination* and related relationships. We discuss design variants for realizing *powertype* structures and compare our approach to alternatives.**

***Index Terms*—multi-level modeling, deep modeling, powertypes, subordination, LML, DLM**

I. Introduction

One of the resultant features of using multiple classification levels for modeling is the possibility to prescribe element properties across more than one level boundary, i.e., the option of using *deep characterization* [5]. Deep characterization can increase the level of control in a model [18] and potency-based deep characterization in particular, supports ensuring consistency and element capabilities in a manner that compares favorably in terms of conciseness to alternatives [17].

One of the reasons for the conciseness of potency-based deep characterization, as used in languages like LML [3] or DLM [16], [23], is the fact that type facets of elements such as Collie and Poodle can be prescribed without the use of a supertype. Compare Figure 1 (a), where an age field is conferred to both Collie & Poodle via a common supertype Dog, to Figure 1 (b) where Collie & Poodle receive the same age^1 field via a potency-two age^2 field in Breed.

The end effect, that any instance of Breed is guaranteed to have an age field, is the same in both scenarios. In case of the potency-based approach (Figure 1 (b)) directly so, and in case of the powertype-based approach (Figure 1 (a)), due to the fact that the "powertype" stereotype on the dependency between Dog and Breed is meant to invoke powertype semantics, i.e., at least the implication that an instance of Breed must specialize Dog (cf. equations D4 and/or D5 of [12]). In other words, the "powertype" relationship establishes a constraint that forces the type facet (here field age) of the supertype, which in this context is also known as the "basetype" (here Dog), upon instances of the type (here Breed). Note that we refrain from referring to Breed as a "metatype" because doing so would only be justified when using absolute terminology. Using relative terminology, Breed is simply Collie's type, i.e., not its "metatype".

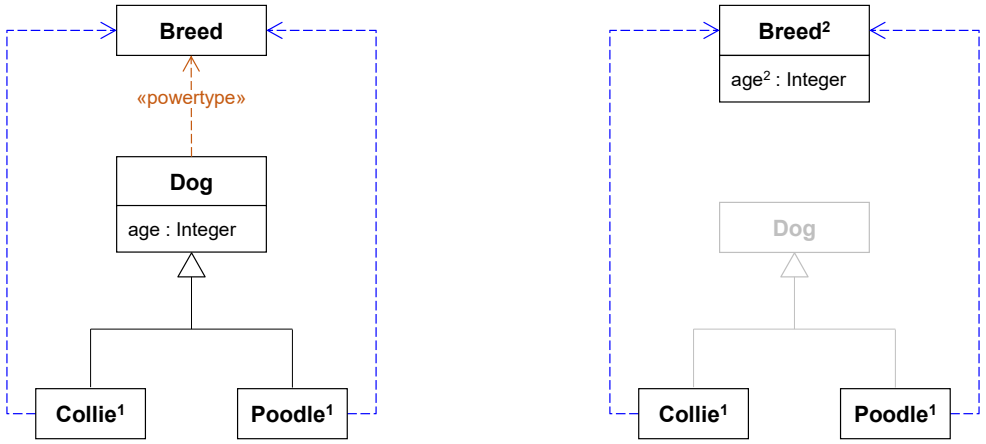
Although the powertype mechanism was not originally conceived as a means to achieve deep characterization (see Section II), languages like Deep Telos [14] or MLT [12], [13], can afford their users with deep characterization without having to introduce any *deep field* concepts (cf. [22]), due to the fact that powertypes can be used to impose type facets on their instances by employing supertypes.

In the following, we shall use a technology independent term to refer to what may be referred to as "powertype instances" or "deep characterizer instances", depending on the deep characterization approach used. Since these elements (here Collie & Poodle) are forced, in one way or another, to partially share a particular type facet, we will refer to them as *constrained types*.

While deep modeling languages do not impose the use of a basetype (like Dog) on their users (cf. [9, section 3.5]), since they can represent the type facet to be shared among the constrained types via a deep type facet (cf. age^2 in Figure 1 (b)), in some cases it is useful to nevertheless introduce a generalization of the constrained types. For example, while a relationship between a dog and its owner could be introduced at the level of Breed via a deep connector [4], it can be more straightforward to model it as an association between Dog and, say Person. Such an association

1) explicitly represents the relationship at its natural level.
2) allows Breed to cover dog and cat breeds while supporting dog ownership and acknowledging that cat ownership is a delusional concept, without having to introduce DogBreed and CatBreed to model the difference.
3) implies a basetype (here Dog) that can naturally host shallow fields (here age).
4) implies that the generalization of all constrained types is made explicit and has a concrete name (cf. [9, Fig. 1(c)]). In Figure 1 (b), for example, there is no information about what Collie and Poodle are meant to generalize to, if the Dog generalization is omitted.

The last aspect may be a downside in some cases, when maximum flexibility is desired and there are, of course, other good reasons to prefer the deep characterizer style in some scenarios [9, Section 3.5]. However, there are also scenarios



(a) Powertype

(b) Deep Characterizer

Fig. 1. Deep Characterization

in which an explicit generalization is welcome. For example, it is common to group constrained types into so-called "generalization sets" [30], which depend on an explicit basetype. In such cases, an explicit generalization (e.g. Dog) then usefully restricts additions, e.g., by clarifying that in our example the constrained types should not be extended by a cat breed.

Indeed, even modelers who have access to the deep characterizer style, sometimes forgo the latter and choose a powertype-like structure instead, in order to

- increase clarity,
- achieve alignment to requirement specifications, and/or
- avoid saddling a deep characterizer with a multitude of deep attributes/connections [10], [19], [20], [25].

The modelers, who produced the models in the cited work, considered the aforementioned advantages worth paying the price for, not only in the form of having to introduce a basetype, but also in the form of establishing powertype semantics through textual constraints (cf. Section V). This suggests that, to support some modeling scenarios, powertype structures should have first-class support in deep modeling languages, even though they are never actually necessary.

In this paper, we first further explore the powertype concept (Section II), then present a proposal on how to support powertypes in deep modeling languages (Section IV), compare the proposal to alternative solutions (Section V), and finally conclude after discussing related work.

II. Background

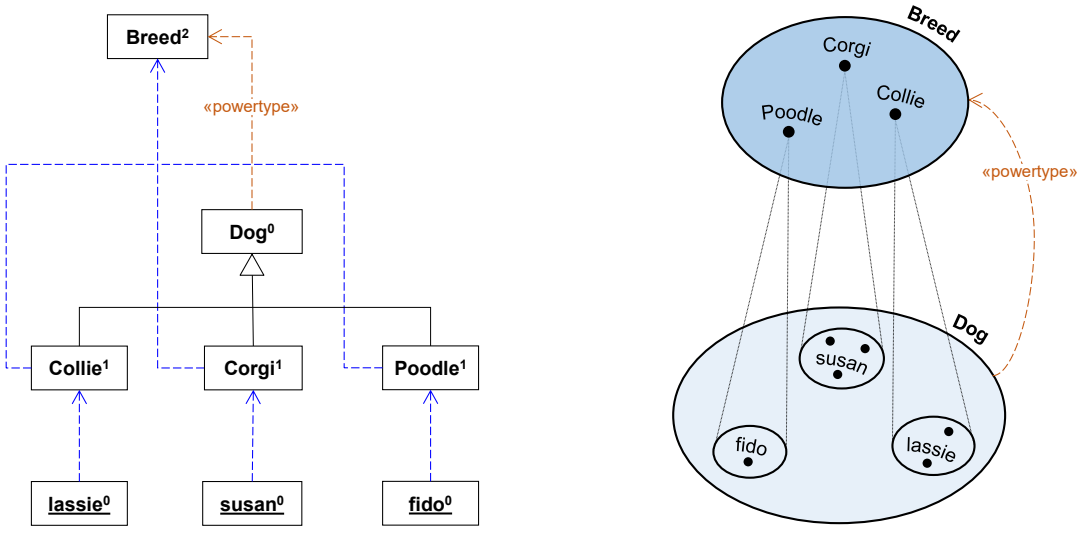
Powertypes were originally not introduced as a means to achieve deep characterization. Odell, for instance, was driven by integrity considerations [29]. He observed that some elements, e.g., Collie, Corgi, and Poodle (cf. Figure 2), are regarded as instances of Breed in some contexts and in entirely separate contexts as subtypes of Dog. He figured that instead of having to maintain consistency between these separate occurrences, i.e., between the constrained types (powertype instances) and the basetype subtypes—e.g., with respect to expanding or reducing the respective element sets—these separate occurrences could be understood as selecting different aspects of the very same elements, i.e., that the constrained types and the basetype subtypes *"…are the same objects."* [28]. Figure 2 (b) depicts this featuring Odell-inspired notation [28], i.e., by using cones to connect the powertype instances (Collie, Corgi, and Poodle), which are members of Breed's extension, with the subsets within the Dog superset, that are extensions of the basetype subtypes.

The "cone" notation is not optimal, as far as we are concerned, since it could be misinterpreted as merely depicting a 1:1 mapping between different entities. To avoid the issue of showing one and the same entity (e.g., Collie) twice, we created Figure 3 using a 3D notation [7]. In this extension-oriented notation, instantiation occurs along the z-axis (upwards). The dual nature of the constrained types (Collie, Poodle, and Corgi) is visualized by them being instances of Breed but sharing their top surfaces with the Dog superset. Note how the superset Dog is not a subset of Breed but hovers one level higher up. It is flat, i.e., not shown to be an instance of anything, because it does not have an ontological type in the model (cf. Figure 2 (a)). Dog has the powertype Breed, but is not an instance of Breed.

By viewing the subtypes Collie, Corgi, and Poodle as instances of Breed (as opposed to maintaining that powertype instances are related to the basetype subtypes but are distinct from the latter), Odell implied two properties:

1) An element like Collie can be an *instance* (of Breed) and a *type* (for collies, such as lassie) at the same time (see Figure 2 (a)). This is obviously at the heart of the "clabject" notion [2].
2) A powertype is a higher-order type; it must be at least a second-order type, given that its instances, the subtypes, are at least first-order types.

Hence, an Odell powertype structure can be justifiably be regarded as an ontological multi-level modeling fragment.



(a) Conceptualization

(b) Extensional view

Fig. 2. Powertype Concept

Note that

- the qualification "ontological" is justified by the fact that a type like Breed (or TreeSpecies as in [29]) does not linguistically classify its instances, as "metamodel elements" typical for that time had done, but rather represents a second-order domain concept [6], [15].
- powertypes can hence not be cleanly supported by two-level technologies.

Numerous modeling languages support powertypes in one way or another. For the purposes of this paper, we focus on UML powertypes as an alternative to Odell powertypes due to their particular characteristics. First, the UML supports powertypes even though it technically only properly supports two levels (objects and their classes) [30]. Similar to UML stereotypes, UML powertypes are supported without being explicitly recognized as higher-order entities. As such, they are not used to define the instance facets of constrained types. Second, they offer another mechanism to modelers that goes beyond what Odell described: a powertype is used as a *discriminator* for a *generalization set* in order to contribute to the latter's properties. A generalization set comprises a subset of basetype subtypes that are delineated by the powertype, for instance BlueCollarWorker and WhiteCollarWorker as subtypes of Worker, with a WorkerType discriminator/powertype. While the subtypes are technically still instances of the powertype, the emphasis is on

- bundling subtypes into coherent subgroups, and
- determining whether such a subgroup is covering and/or disjoint [30, Section 7.3.21].

A generalization set with the properties "covering" and "disjoint" can be referred to as a *partition* (of the basetype). In Figure 2, this would imply that every dog must be exactly of one of the three listed breeds, i.e., the extension of Dog would have no member outside any of the three subsets shown in Figure 2 (b), and the subsets would be prevented from overlapping.

A UML powertype is therefore probably best understood as a derived concept, i.e., by starting with the subsets implied by basetype subtypes, making a selection among those subsets based on whether the instances within the subsets can be separated by the same discrimination principle (here, dog breed)—in other words, determining the generalization set constituents—and finally naming the discrimination principle accordingly (here, DogBreed or Breed), meaning that the subsets in Figure 2 (b) come first with Breed being a derived classifier.

This is in contrast to powertype structures being used in multi-level modeling languages to achieve deep characterization, as in this application the powertype exists first, an instance (say Collie) is created, and the latter is then forced to specialize the basetype (here Dog). Moreover, in many multi-level modeling languages/formalisms, the question of whether or not the constrained types are covering and/or disjoint is not of concern (with MLT being one of the exceptions [12]).

Finally, in a deep characterization scenario the primary objective is not to group certain subtypes and/or let them share a common type facet, rather the objective is to ensure that all instances of the constrained types are guaranteed to have certain properties (which are stipulated by the basetype). This is the reason why Figure 1 (b) is captioned *"Deep Characterizer"*, since the ultimate objective is not to endow both Collie and Poodle with a potency-one field, but rather to ensure that all collies and all poodles have an "age" property.

Given the aforementioned properties of powertype structures when used for deep characterization, the question arises which powertype semantics should be adopted.

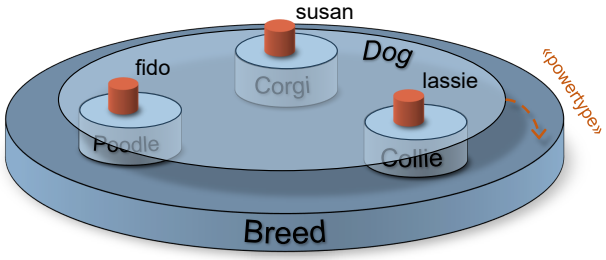


Fig. 3. Type vs Supertype

III. Suitable Semantics

In the following, we use "$e_i : e_t$" to represent that "e_i is a direct instance of e_t" and "$e_{sub} \prec e_{super}$" to represent that "e_{sub} specializes e_{super}". Carvalho et al. [12] define their "`isPowertypeOf`" relation as follows[1]:

$$\begin{aligned} \forall\, e_b, e_p \quad & \text{powertypeOf}(e_p, e_b) \equiv \\ & \forall\, e_s \;\; e_s : e_p \;\leftrightarrow\; e_s \prec e_b \end{aligned} \qquad (\text{P}_{MLT})$$

This may be read as

> *Constrained types (e_s) specialize a basetype (e_b) that is associated with a powertype (e_p), if and only if they are instances of said powertype.*

This means that all subtypes of the basetype must be instances of the powertype, i.e., in the example of Figure 2, one would not be able to add a subtype Mongrel as a specialization of Dog, since such a subtype would not satisfy the requirements for being a dog breed. For the purposes of achieving deep characterization, this powertype variant would therefore be unnecessarily strict, since one is only interested in constraining instances of the powertype (Breed) to be subtypes of the basetype (Dog). In other words, one is not concerned with elements that are not instances of the powertype and it should not be an issue to have subtype Mongrel specializing Dog.

The UML clearly supports the latter requirement with its powertype approach, which can be expressed as follows:

> *All constrained types (e_s) that are members of a generalization set (g_s), must be instances of the generalization set's powertype (e_p), and must furthermore specialize the associated basetype (e_b).*

Formally:

$$\begin{aligned} \forall\, e_b, e_p \quad & \text{powertypeOf}(e_p, e_b) \equiv \\ & \exists\, g_s \;\; \forall\, e_s \\ & e_s \in g_s \;\rightarrow\; (e_s : e_p \wedge e_s \prec e_b) \end{aligned} \qquad (\text{P}_{UML})$$

[1]We slightly renamed it to "powertypeOf" and, for clarity, omitted a term whose only purpose is to prevent the relation from being vacuously true, in case the set of constrained types is empty.

Given that a generalization set has an associated powertype—

$$e_s \in g_s \;\leftrightarrow\; e_s : e_p,$$

—the condition boils down to

$$e_s : e_p \;\rightarrow\; e_s \prec e_b,$$

which in turn means that a UML powertype is like an MLT powertype (see Equation P_{MLT}), with the difference being that the implication only goes one way, i.e., there is no need for all subtypes of the basetype to be an instance of a powertype.

As a result, a UML powertype is clearly suited to support deep characterization without imposing unnecessary constraints on basetype subtypes. However, as mentioned in Section II, if one is not concerned with anything else but deep characterization, it is not necessary to introduce the notion of a generalization set. Rather, Odell's initial take is sufficient:

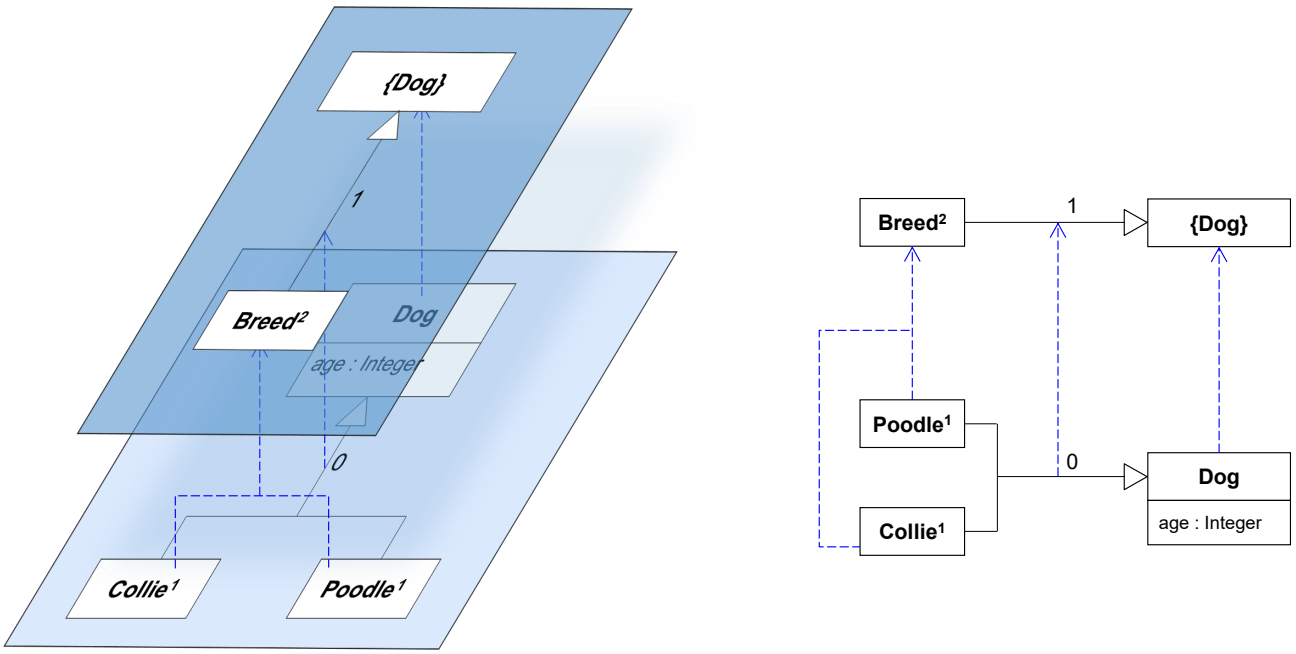
> *"A power type is a type whose instances are subtypes of another type"* [28, p. 252]).

—provided that a reasonable interpretation is applied: Odell's description technically allows more than one basetype, since *"another type"* is not guaranteed to be the same (base-)type for every subtype. This would make the definition unfit for deep characterization applications. However, based on the examples presented by Odell, he most likely meant *"of one shared type"* rather than *"of another type"*. With that interpretation, Odell's powertype can be formalized as:

$$\begin{aligned} \forall\, e_b, e_p \quad & \text{powertypeOf}(e_p, e_b) \equiv \\ & \forall\, e_s \;\; e_s : e_p \;\rightarrow\; e_s \prec e_b \end{aligned} \qquad (\text{P}_{Odell})$$

This definition shares the same spirit with an UML powertype, without having to resort to a generalization set concept. Carvalho et al. label this powertype variant "`categorizes`" [12].

Note that the "$\equiv$" in Equation P_{Odell} means that if two types P and B are declared to be participants of a powertype relationship (see the dependency between Dog and Breed in Figure 2 (a), with the "powertype" stereotype), then the right-hand side "$\equiv$" establishes the desired constraint on powertype (P) instances to be required to specialize the basetype (B).



(a) Iso-Level 3D

(b) Iso-Level 2D

Fig. 4. Deep Specialization

IV. Design Alternatives

In the following, we look at ways of supporting the semantics of Odell-powertypes / MLT-categorization without having to introduce a dedicated powertype construct (see Section V-B for a respective trade-off analysis).

To this end, we leverage already existing language mechanisms in LML and DLM. The key idea is to regard the specialization relationship between two types as a potency-zero connection (aka "link") between those types. If one then allows such potency-zero connections to be classified, i.e., introduces the notion of *higher-order specializations*, and allows the latter to use mandatory multiplicities, it becomes possible to stipulate the presence of (direct or indirect) specialization relationships between types, just like associations may be used to enforce the presence of links between objects.

Since the purpose of a powertype relationship, at least when used for deep characterization, ultimately is to enforce, for all constrained types, the presence of a (direct or indirect) specialization relationship to a shared basetype, using specialization types with mandatory multiplicities is a viable alternative. Below, we consider two variations of the same idea and then look beyond supporting powertypes only.

A. Iso-Level Relationship

The variant that is most straightforward to explain uses a higher-order specialization between two elements that are located on the same level. Figure 4 (a) depicts the idea, using a subset of the model from Figure 1 as an example. Ignoring multiplicities for the time being, we labeled the specialization relationships between the constrained types and the basetype (here Dog) as potency-zero links (see the bottom part of Figure 4 (a)).

All these specialization relationships are instances of the single potency-one specialization between Breed and {Dog} at one level higher up. Here, {Dog} denotes a singleton type whose only instance is Dog. The use of such a singleton type is necessary to ensure that all instances of Breed entertain potency-zero specialization relationships to the same basetype.

Figure 4 (b) shows the same scenario as Figure 4 (a), just using conventional notation. Note that Breed^2 does not specialize {Dog}. Only if the relationship between those two elements were a potency-zero specialization, a specialization relationship would be established between them. The actual potency-one specialization is akin to a *specialization association*. It is debatable whether different concrete syntax should be used for higher-order specializations to avoid potential confusion, but for now we trust that users will be able to manage. After all, UML associations and links share the same concrete syntax as well.

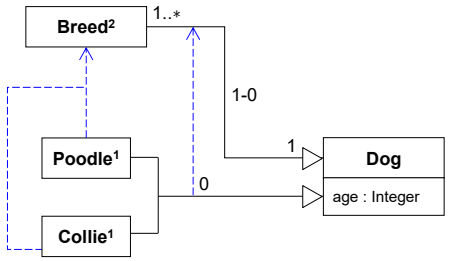
The following trade-offs apply to the above described iso-level deep (i.e., higher-order) specialization approach:

+ existing language features (cf. [4]) are leveraged to forgo the introduction of a further language construct.

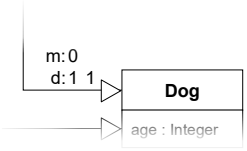
+ choice over multiplicities suggests that more than just powertype relationships may be captured.

+ compatible with the "strict metamodeling" doctrine.

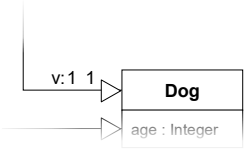
– in comparison to a regular powertype structure, an additional singleton type is required, and some language support for defining a singleton type in terms of the one instance it allows is required.



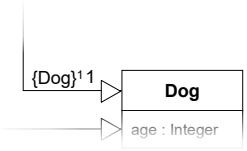
(a) Dual Deep Instantiation



(b) Mutability and Durability



(c) Value Potency



(d) Relationship End Restriction

Fig. 5. Cross-Level Alternatives

B. Cross-Level Relationship

In order to address the downside of the iso-level variant, it is necessary for the powertype (Breed) to refer to basetype (Dog) itself, rather than to a de facto proxy ({Dog}). Figure 5 (a) shows a cross-level variant of the scenario shown in Figure 4 (b), which uses the *dual potency* mechanism by Neumayr et al. [27]. The dual potency specification "1–0" for the deep specialization between Breed and Dog denotes that upon instantiation, the Breed-end of the relationship (with potency "1") will be connected to a breed instance (e.g. Collie), while the generalization-end (here the Dog-end) of the specialization (with potency "0") will be retained. Note that at the Breed-end, the deep specialization prescribes, via a multiplicity specification, that at least one constrained type must specialize Dog. The "mandatory"-multiplicity ("1") at the generalization end (here, the Dog-end) specifies that a constrained type must specialize exactly one supertype.

The two instantiated specializations between the constrained types and Dog do not feature multiplicity specifications because they are not types for any lower-order specializations. Although we show a zero potency value for these specializations here for the sake of explaining the scenario, we suggest to normally not show zero potency values for specializations.

Figure 5 (b) shows the same cross-level design, but focuses on the deep specialization's generalization end (Dog-end), replacing the dual potency approach of Figure 5 (a) by a specification of the "mutability" and "durability" values (cf. [3]) of the generalization end. This variant highlights the fact that the generalization end's (type) value can be understood as a *regularity attribute* [1]. A regularity attribute is characterized by instances (here the generalization-ends of the potency-zero specialization relationships between the constrained types and Dog) sharing the same value (here "Dog") for a property which is specified at one level higher up (here the Dog-end of the deep specialization between Breed and Dog).

The variant shown in Figure 5 (c) effectively establishes the same specification but employs the notion of *value potency* [8]. The standard notion uses a potency on the value itself—e.g., as in "taxRate^{1} = 19%^{1}"—but since that could be confused with a clabject potency if applied to Dog, we chose a less ambiguous notation. We are not suggesting that this notation is optimal; we are merely communicating the various alternatives available for restraining the generalization end value of the potency-zero specialization relationship to a specific element.

Although the approaches shown in Figures 5 (b) & (c) already communicate that the relationship end value Dog (to be used for all potency-zero specializations) is prescribed at the type level via type-level constraints, perhaps Figure 5 (d) most directly depicts that the relationship end value (here Dog) to be used for all potency-zero specializations at the level below, is prescribed at the type level (here via {Dog}). The "{...}" set notation is meant to denote a restriction of the values that may be chosen for that specialization end upon instantiation of the deep specialization (remotely related, but distinct from the "redefines" approach for association ends in the UML). This means that upon instantiation of the potency-one specialization between Breed and Dog, only Dog can be used as a participant at the right-hand side. This variant shares with the approach shown in Figures 5 (b) & (c) that the deep specialization end has (durability) potency one, but it replaces the specification of an immutable value (Dog), that is supposed to be transferred upon instantiation of the deep specialization, with a type-level value ({Dog}) which only makes one value available for instantiation.

The following trade-offs apply to the above described cross-level deep specialization approaches:

+ all positive aspects of the iso-level approach apply, with the exception of strict metamodeling compatibility.

+ no need to explicitly introduce a singleton type.

– requires one of the supporting mechanisms used in Figures 5 (a)–(d).

– no "strict metamodeling" compatibility.

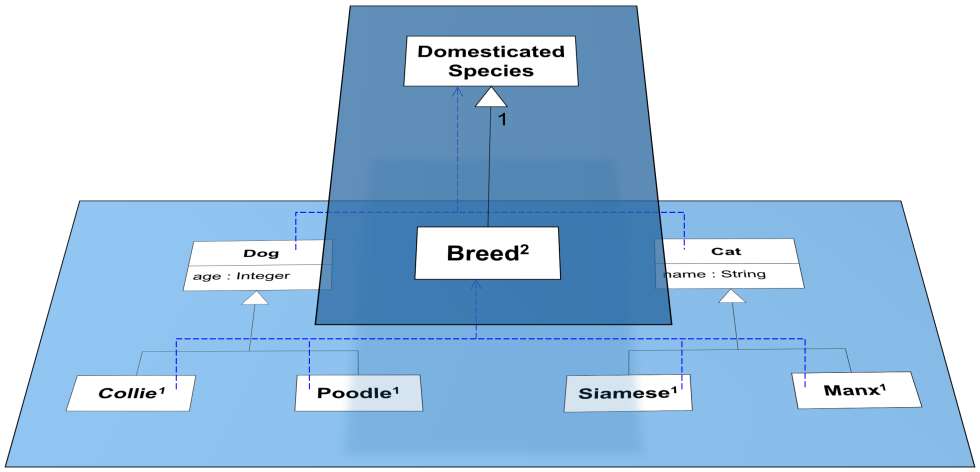


Fig. 6. Subordination

Since the cross-level variant is a bit more parsimonious, the question which of the above two alternatives is preferable is hence most likely dictated by whether or not strict metamodeling needs to be adhered to.

Interestingly, however, deep specialization has more to offer than just providing a way to stipulate powertype structures.

C. Generalization

Given that specifying powertype structures using deep specialization relies on specific restrictions—regarding which and how many values are allowed at the generalization end of a specialization relationship—it is obvious that deep specialization should support other modeling requirements.

Consider Carvalho et al.'s definition of the "*subordination*" relationship (slightly simplified, analogous to the footnote on Equation P_{MLT}) [12, (D3)]:

$$\begin{aligned} \forall\, t_1, t_2 \quad & \mathrm{subordinateTo}(t_1, t_2) \equiv \\ & \forall\, e_1 \;\; e_1 \!:\! t_1 \;\; \rightarrow \;\; (\exists\, e_2 \;\; e_2 \!:\! t_2 \wedge e_1 \prec e_2) \end{aligned} \tag{Sub}$$

This may be read as

> *A metatype (t_1) is subordinate to another metatype (t_2), if and only if every instance of the former specializes an instance of the latter.*

For example, one could require that every dog breed like Collie & Poodle, i.e., an instance of Breed, must specialize an instance of DomesticatedSpecies, e.g., Dog, by making Breed subordinate to DomesticatedSpecies. Other breeds, such as the cat breeds Siamese & Manx, then would also have to specialize an instance of DomesticatedSpecies, e.g., Cat, see Figure 6. For simplicity, we leave the multiplicity at the Breed-end unspecified and do not explicitly show that the potency-zero specialization relationships at the lower level are instances of the deep specialization between Breed and DomesticatedSpecies. Note the multiplicity of "1" though, at the generalization end of that deep specialization, which ensures a single basetype per subordination incarnation.

Note that in the example, subordination

1) does not imply specialization between Breed and DomesticatedSpecies. Subordination only constrains instances of the subordinated type to specialize instances from the subordinating type.
2) can be regarded as a generalization of "powertyping". Instead of specifying one concrete basetype (e.g. Dog), one only specifies the type of such a basetype (here DomesticatedSpecies), and leaves open the possibility of multiple basetypes (cf. Dog & Cat in Figure 6).

This means "powertyping" is subsumed by subordination since one may express powertyping as a constrained form of subordination, using a singleton type in the "subordinating" role:

$$\begin{aligned} \forall\, e_p, e_b \quad & \mathrm{powertypeOf}(e_p, e_b) \equiv \\ & \mathrm{subordinateTo}(e_p, \{e_b\}) \end{aligned} \tag{Gen}$$

Compare Figures 6 & 4 (a) noting the difference between DomesticatedSpecies and {Dog}, analogue to Equation Gen.

Note that placing a multiplicity constraint like "2..3" at the Breed-end of the deep specialization and removing the "1" multiplicity at the other end, would allow one to specify that whenever a domesticated species is introduced that it must have two to three subtypes. By declaring the potency of DomesticatedSpecies to be "1", one could even force subtypes to be *covering* with respect to their basetype since then DomesticatedSpecies instances are forced to have potency zero, i.e., lack the ability to have any direct instances of their own. A type representing a duality, such as "chirality" could thus be forced to have exactly two subtypes that cover both cases. Entirely capturing the notion of an exclusive duality, however, would require ruling out cases in which an instance is classified by more than one subtype; which is a kind of constraint that we do not aim to cover in this work. See the discussion of Carvalho et al.'s "≪instantiation≫" stereotype in Section VI for an outlook as to how such a constraint could be established.

V. Comparison

Since our main objective in this paper is to use deep specialization as a means to realize powertype structures, in the following, we are discussing its merit compared to respective alternatives, i.e., using constraints or a dedicated powertype construct. We do not discuss characteristics that are common to all these approaches but rather focus on the differences.

A. Constraint

In [19], both solutions feature DOCL constraints in order to enforce powertype structures [24]. Listing 1, taken from the Melanee solution, shows how instances of HuaweiMPModel are required to specialize HuaweiMPDevice. The constraint, therefore, effectively connects HuaweiMPDevice in a basetype role to HuaweiMPModel in a powertype role.

```
context HuaweiMPModel(1_1)
inv: self.#getSuperTypes()# -> collect(#name#)
     -> includes("HuaweiMPDevice")
```

Listing 1. Powertype constraint example

The trade-offs of this powertype realization alternative are:

+ no mechanism/concepts beyond constraints are required. Since constraints have many other valuable applications, it is likely that constraint support is available anyhow.

+ the constraint is straightforward to write/read for those familiar with constraint languages, and its structure is not domain-specific.

– model users unfamiliar with constraints in general, or just the particular constraint language used, may find it difficult to understand what the constraint is about.

– a textual constraint is typically not integrated with the diagram containing the main model content and may thus be somewhat hidden from view. Even visual representations of such a constraint [31], if available at all, are unlikely to be part of regular model diagrams.

– without advanced tool support that links the names of model elements in a constraint (here "HuaweiMPModel" & "HuaweiMPDevice") to model elements in a diagram, model maintenance becomes an issue. Any changes to the model, e.g., a simple renaming of involved elements, would have to be followed by updates to any affected constraints.

B. Dedicated Construct

Languages like UML [30], MLT [12], and DeepTelos [14] use relationships specifically designed to express a powertype relationship, i.e., they are custom-tailored to establish no more and no less than powertype structures. While implementation details differ, it is fair to say that all the above named and other similar approaches effectively amount to having a dedicated language construct, or at least specific library vocabulary, available. The trade-offs of this approach are:

+ direct communication regarding the purpose of the construct/relationship.

+ no need to pay attention to parameters, or similar details, to figure out the exact meaning.

+ given the affinity to the instanceOf relationship (elements with an order difference of one are connected) a powertype relationship would be compatible with languages like LML that embrace the "strict metamodeling" paradigm [2], even though it crosses a metalevel boundary.

– modelers need to be familiar with the semantics of the dedicated support. Since the latter is not notated using common notions such as classification or specialization, modelers need to be cognizant of the powertype concept. While many modelers can be expected to be familiar with powertypes, modeling beginners should receive consideration as well.

– a powertype construct serves exactly one purpose; it is not clear how related structures such as partitions, subordinations, etc. could be subsumed by it, meaning that the latter probably require their own dedicated support.

– indiscriminate addition of dedicated language constructs may result in undue language complexity growth.

C. Deep Specialization

As alluded to earlier already, the notion of *deep specialization* is automatically supported by a language that supports deep connections [4], as long as specialization relationships are regarded as potency-zero links that can be classified. However, even just two-level support with respect to specialization relationships would suffice to cover the applications touched upon in this paper, including support for subordination (cf. Section IV-C).

The trade-offs of deep specialization, therefore, are:

+ no extra language construct required.

+ modelers do not need to know the semantics of powertype constructs. Instead, the semantics are explicitly spelled out in the form of higher-order specialization. Anyone familiar with the semantics of associations can figure out what the intended constraints are.

+ applicable in both strict and non-strict frameworks (cf. Figure 4 vs Figure 5).

+ applicability exceeds pure powertyping. A range of other constraints on model structures can be easily established via respective multiplicity specifications and/or relationship end value restrictions.

+ deeply specifying the presence of powertype structures across more than one level boundary, or as a matter of fact, other structures such as subordination, is naturally supported via multiplicity specifications with potencies other than "1" (cf. [4]).

– semantically different constraints are expressed using the same main syntax and are merely distinguished by multiplicity choices. Modelers, hence, need to pay attention to detail but potentially may develop respective pattern recognition, similar to that in effect when certain association multiplicities are effortlessly categorized as "'optional", "mandatory", etc.

– depending on concrete syntax choices, deep specializations may be confusing for those unfamiliar with the notion. Retaining the notation of potency-zero specializations for higher levels, as we did in this paper, may result in misinterpretations. An alternative notation or some means of emphasizing the difference between regular potency-zero specialization and higher-order specialization may be advantageous in practice.

One may add that diagrams simultaneously involving deep classification and deep specialization present layout challenges, however, the same applies to the use of visual representations of dedicated powertype relationships.

There are various trade-offs that depend on which of the deep specialization flavors in Section IV are chosen, i.e., which particular kind of deep specialization end specification mechanism is chosen (see Figures 5 (a)–(d)). Due to space constraints, we cannot cover them here.

VI. Related work

Cardelli was one of the earliest authors to discuss the notion of powertypes [11]. However, his powertype notion is based on the mathematical notion of a powerset and is therefore not well-suited in constructive modeling approaches featuring nominal types. The latter feature powertypes whose instances correspond to a very small subset of a powerset.

Deep characterization and powertype structures ultimately achieve identical object specifications and since any basetype properties can be lifted to the powertype—provided deep field and/or deep connection support is available—converting between these two alternatives should be a refactoring [26]. However, this is only the case for the asymmetric powertype definition we chose (see EquationP_{Odell}), since a deep characterization approach would not imply the symmetric definition of Equation P_{MLT}.

Carvalho et al. recommend using an "≪instantiation≫" stereotype to mark an association between the basetype and the powertype as being distinguished "*…from other domain relations that do not have an instantiation semantics*" [13, p. 7]. Using various multiplicity combinations on that association allows them to specify whether generalization sets are complete and/or disjoint [13, Table 2]. This is analogous to the expressiveness afforded by using various multiplicity combinations on deep specializations. However, note that the "≪instantiation≫" association targets potency-zero instanceOf links whereas deep specialization targets potency-zero specialization links, which makes them complementary control mechanisms.

Note that the "complete"/"covering" aspect of MLT's control—i.e., specifying that the basetype may not have any instances which are not also instances of at least one of the constrained types—is naturally supported in our approach by forcing the basetype's potency to be zero from the level above. However, control over whether the constrained types are "overlapping" or not, would require treating instanceOf relationships as links whose outdegrees can be controlled via higher-order instantiation relationships that are part of the user model, similar to Carvalho et al.'s approach. The question of whether or not such control should be added to deep modeling languages is out of scope for this paper.

Carvalho et al. demonstrate MLT's support for "`partitions`" and "`isSubordinateTo`" relationships using a biological taxonomy [12, Fig. 12]. With our approach, those relationships—plus the reflexive association "isSubordinateTo", which is distinct from proper "`isSubordinateTo`" relationships—could to a large extent be expressed with respective deep specializations. It should be noted that we currently do not fully support modeling "complete categorizations" ("covering constrained types" in our terminology) and "partitions" with respect to a particular powertype. Furthermore, at this point in time we do not support MLT's cross-level relationships "`isPowertypeOf`" and "`disjointlyCategorizes`" either [12, Table 2].

DeepJava considered the Java *extends* relationship to have potency zero and used potency-one "*extends*[1]" to effectively establish a powertype relationship [21, Listing 8]. This work did not touch upon using potencies or multiplicities at *extends* relationship ends though.

VII. Conclusion

Deep characterization is an important aspect of multi-level modeling and while deep instantiation may realize it in the most parsimonious manner, its extreme efficiency is not always desirable. Powertype structures are therefore not only useful when languages do not support other means of achieving deep characterization, but can also improve modeling pragmatics when using deep modeling languages such as LML and DLM.

After having analyzed various variants of powertype concepts, we considered two main variants—strict iso-level deep specialization vs cross-level deep specialization—to support the enforcement of powertype structures without requiring anything else but shallow classification support for specialization relationships. Viewing the latter as potency-zero *links* allowed us to leverage the known modeling notions of specialization and multiplicities to realize powertype support without introducing an additional language construct.

The fact that we could identify four alternative ways to support cross-level deep associations appears to give credence to the notion that the restrictions necessary to establish powertype structures on the basis of higher-order specializations are rather natural. If the need to restrict the number of, and type of, values available to choose from for a relationship end in the form necessary for specifying a powertype relationship had been obscure, we would not have been able to identify so many already existing solutions for it. Furthermore, the number of alternatives available makes it likely that one of them is already supported in a host language.

We maintain that avoiding language construct bloat and the self-evident semantics of deep specializations are aspects very much worth considering when designing languages. Our notational choices for higher-order specialization surely would benefit from validation, but the fact that they can visually convey powertype semantics, without requiring modelers to reference formal definitions, seems very promising to us.

Even the generalization of "powertyping" to subordination does not require an external definition, it can rather easily be captured via an appropriate use of a multiplicity specification at the generalization end of a higher-order specialization. Embracing higher-order specification in the form of deep specialization thus not only makes it possible to explain advanced

relationships like subordination in terms of known multi-level modeling concepts, but also provides a means for the aforementioned use of notation with self-evident semantics.

Obviously, there is much left to explore with respect to useful combinations of multiplicity specifications and, in particular, deep specializations with potencies higher than "1", featuring several multiplicity specifications per relationship end, each with a different potency value.

In this paper, we did not aspire to completely cover the range of associated specification options regarding disjoint generalization sets, as supported by the UML and MLT, for instance. This would be important if one wanted to comprehensively specify generalization sets, however, in this paper, we purely focused on the deep characterization aspect of powertypes.

Although we believe that our deep specialization proposal to capture powertype structures stands on its own and should be able to provide practical value to deep modeling users, perhaps a wider reaching contribution is the observation that "powertyping" is a special case of subordination (where the subordinating type is restricted to an anonymous singleton type) and that both these important relationships can be captured via higher-order specializations.

References

[1] J. P. A. Almeida, V. A. Carvalho, C. M. Fonseca, and G. Guizzardi. A note on properties in multi-level modeling. In *2021 ACM/IEEE Int Conf Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 497–501. IEEE Computer Society Press, 2021.

[2] Colin Atkinson. Meta-modeling for distributed object environments. In *Enterprise Distributed Object Computing*, pages 90–101. IEEE, October 1997.

[3] Colin Atkinson and Ralph Gerbig. 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, 2016. Köllen.

[4] Colin Atkinson, Ralph Gerbig, and Thomas Kühne. A unifying approach to connections for multi-level modeling. In *Proceedings of MODELS'15*, pages 216–225. IEEE Computer Society, 2015.

[5] Colin Atkinson and Thomas Kühne. Rearchitecting the uml infrastructure. *ACM Transactions on Modeling and Computer Simulation*, 12(4):290–321, October 2002.

[6] Colin Atkinson and Thomas Kühne. Demystifying ontological classification in language engineering. In *Modelling Foundations and Applications*, volume LNCS 9764, pages 83–100. Springer, 2016.

[7] Colin Atkinson, Thomas Kühne, and Brian Henderson-Sellers. To meta or not to meta – that is the question. *Journal of Object-Oriented Programming*, 13(8):32–35, December 2000.

[8] Colin Atkinson, Thomas Kühne, and Brian Henderson-Sellers. Systematic stereotype usage. *Journal on Software and Systems Modeling*, 2(3):153–163, 2003.

[9] Colin Atkinson, Thomas Kühne, and Arne Lange. Misconceptions about potency-based deep instantiation. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 810–817, New York, NY, USA, 2024. Association for Computing Machinery.

[10] Sandor Bacsi, Arne Lange, Thomas Kühne, Gergely Mezei, and Colin Atkinson. Melanee and DMLA – A Contribution to the MULTI 2021 Collaborative Comparison Challenge . In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 556–565, Los Alamitos, CA, USA, October 2021. IEEE Computer Society.

[11] Luca Cardelli. Structural subtyping and the notion of power type. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 70–79, San Diego, California, 1988.

[12] Victorio A. Carvalho and João Paulo A. Almeida. Toward a well-founded theory for multi-level conceptual modeling. *Software & Systems Modeling*, 17(1):205–231, 2016.

[13] Victorio Albani Carvalho, João Paulo A. Almeida, and Giancarlo Guizzardi. Using a well-founded multi-level theory to support the analysis and representation of the powertype pattern in conceptual modeling. In Selmin Nurcan, Pnina Soffer, Marko Bajec, and Johann Eder, editors, *Advanced Information Systems Engineering*, pages 309–324, Cham, 2016. Springer International Publishing.

[14] Manfred A. Jeusfeld and Bernd Neumayr. DeepTelos: Multi-level modeling with most general instances. In *Conceptual Modeling - 35th International Conference, ER 2016*, pages 198–211, 2016.

[15] Thomas Kühne. Matters of (meta-) modeling. *Software and System Modeling*, 5(4):369–385, 2006.

[16] Thomas Kühne. Multi-dimensional multi-level modeling. *Software and Systems Modeling*, 21(2):543–559, 2022.

[17] Thomas Kühne and Colin Atkinson. Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359, Jul 2008.

[18] Thomas Kühne and Arne Lange. Meaningful metrics for multi-level modelling. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[19] Thomas Kühne and Arne Lange. Melanee and dlm: a contribution to the multi collaborative comparison challenge. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '22, page 434–443, New York, NY, USA, 2022. Association for Computing Machinery.

[20] Thomas Kühne and Pierre Maier. Fmmlx and dlm – a contribution to the multi collaborative comparison challenge. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 800–809, New York, NY, USA, 2024. Association for Computing Machinery.

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

[22] Thomas Kühne, João Paulo A. Almeida, Colin Atkinson, Manfred A. Jeusfeld, and Gergely Mezei. Field types for deep characterization in multi-level modeling. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 639–648, 2023.

[23] Thomas Kühne and Manfred A. Jeusfeld. Supporting sound multi-level modeling—specification and implementation of a multi-dimensional modeling approach. *Data & Knowledge Engineering*, 160:102481, 2025.

[24] Arne Lange. dACL: the deep constraint and action language for static and dynamic semantic definition in Melanee. Master's thesis, University of Mannheim, 2016.

[25] Arne Lange and Colin Atkinson. Multi-level modeling with LML – A contribution to the multi-level process challenge. *International Journal of Conceptual Modeling*, 17, 2022. Special Issue: Multi-Level Process Challenge.

[26] Juan De Lara and Esther Guerra. Refactoring multi-level models. *ACM Trans. Softw. Eng. Methodol.*, 27(4), November 2018.

[27] Bernd Neumayr, Christoph G. Schuetz, Manfred A. Jeusfeld, and Michael Schrefl. Dual deep modeling: multi-level modeling with dual potencies and its formalization in f-logic. *Software & Systems Modeling*, 17(1):1–36, 2016.

[28] James Odell. *Object-Oriented Analysis and Design*. Prentice-Hall, 1992.

[29] James Odell. Power types. *Journal of Object-Oriented Programming*, 7(2):8–12, May 1994.

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

[31] Edward D. Willink. Ocl visualization - a reality check. In *19th International Workshop in OCL and Textual Modeling (OCL 2019)*, volume Vol-2513, pages 17–30. CEUR Workshop Proceedings, 2019.