Melanee and DLM – A Contribution to the MULTI Collaborative Comparison Challenge

Conference Paper · October 2022

CITATIONS
0

READS
4

2 authors:

Thomas Kühne
Victoria University of Wellington
96 PUBLICATIONS **3,001** CITATIONS
SEE PROFILE

Arne Lange
Universität Mannheim
8 PUBLICATIONS **19** CITATIONS
SEE PROFILE

Melanee and DLM

A Contribution to the MULTI Collaborative Comparison Challenge

Thomas Kühne
thomas.kuehne@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

Arne Lange
lange@informatik.uni-mannheim.de
University of Mannheim
Mannheim, Germany

ABSTRACT

Responding to the MULTI 2022 Collaborative Comparison Challenge, whose purpose is to promote cross-fertilization between multi-level modeling approaches, we compare Melanee- and DLM-based solutions. We first present each approach and solution separately, and then discuss their strengths and weaknesses.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**; • **Computing methodologies → Modeling methodologies**.

KEYWORDS

multi-level modeling, modeling challenge, Melanee, DLM

ACM Reference Format:
Thomas Kühne and Arne Lange. 2022. Melanee and DLM: A Contribution to the MULTI Collaborative Comparison Challenge. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3550356.3561571

1 INTRODUCTION

Multi-level modeling aims to minimize accidental complexity by allowing modelers to represent concepts from the subject of interest at the optimal level of abstraction using an unbounded number of metalevels [6, 20]. As the number and variety of multi-level modeling approaches has increased over recent years, so has the debate about what exactly multi-level modeling is and what its core principles are. To foster this debate, the multi-level modeling community has issued so-called "challenges" which were designed to advance the discipline by facilitating analyses of how different approaches model the same domain properties and requirements [1].

The most recent such challenge in the MULTI series of workshops is the *Collaborative Comparison Challenge*. It invites multi-level modelers to not only present their own solutions to the challenge but also to collaborate with another research group to provide a deeper discussion of their approaches' respective properties [25]. This paper, prepared by researchers responsible for the principles underlying the Melanee [3] and DLM (Domain Level Modeling) [17, 19] multi-level modeling technologies, is a response to the 2022 call for collaboration comparison challenge submissions.

In this paper we first provide brief characterizations of Melanee and DLM in section 2, then present respective solutions to the domain challenge in section 3, and subsequently analyze the key commonalities and differences between the approaches, providing a discussion of the respective pros and cons in section 4. Section 5 concludes the paper.

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10…$15.00
https://doi.org/10.1145/3550356.3561571

2 MODELING APPROACHES

In this section, we characterize the two multi-level modeling approaches we used to model our solutions.

2.1 Melanee

Melanee [16] is based on a flavor of multi-level modeling, referred to as "deep modeling" [4], in which ontological and linguistic classification relationships are separated into different dimensions, and the ontological classification levels mirror domain classification levels. Since model elements can influence lower-level elements across multiple levels through so-called *deep characterization*, the complete multi-level model encompassing all domain information about a subject is referred to as a "deep model" and each ontological level within this deep model is referred to as a "level".

In Melanee, deep models obey the principles of "strict metamodeling" [2] and are represented using the Level-agnostic Modeling Language (LML). They can be augmented with a variant of OCL called "DOCL" [21], which is enhanced to be aware of, and build on, deepness. The LML defines three core constructs – entities, connections and generalizations – with entities corresponding to classes and/or objects in classic UML-style structural modeling and connections corresponding to associations (including association classes) in the UML, rendered using a slightly different notation. Connections can be navigable or non-navigable and can capture two forms of containment, composition and aggregation, using a notation similar to the UML. Entities and connections are clabjects which have a "potency" that specifies over how many levels they can be instantiated. Finally, the generalization relationship is used to represent inheritance using the unfilled-triangle notion of the UML [16].

Potency is the most fundamental of three so-called "vitality" properties in LML and specifies the instantiation depth of a clabject [6, 22], i.e., the length of instantiation chains that may originate from a clabject. It is captured as an integer-valued property of a clabject and adheres to two fundamental rules:

(1) the potency of a clabject cannot be less than 0.
(2) the potency of a direct instance of a clabject must be less than the potency of that clabject [17].

The two other vitality attributes are "durability" and "mutability", which govern the existence of ontological clabject attributes and the values of those attributes, respectively. Durability is a property of the ontological attributes of clabjects which characterizes their endurance over instantiation steps. Like potency, it is represented by a non-negative integer, but unlike clabject potency, attribute durability must be exactly one less than the durability of its attribute type. Instances of a clabject that has a durability-zero attribute do not have an instance of that attribute. Mutability, in contrast, is a non-negative integer property of the ontological attributes of clabjects that characterizes how their values can change over instantiation steps. A zero mutability value means that the instances of an attribute must have the same value. The rules for mutability are essentially the same as those for durability except that if the mutability of an attribute is zero, the mutability of instances of that attribute must also be zero.

2.2 DLM

Domain Level Modeling (DLM) originated from joint work that also gave rise to Melanee/LML. Hence there are a number of fundamental similarities between DLM and Melanee, such as being based on the orthogonal classification architecture (OCA) [8], being a deeply characterizing "classification potency"-based [6, 17] "deep modeling" approach [4], and featuring an order-alignment level segregation principle [18].

Unlike Melanee, which also aims to support the definition and use of domain-specific languages [16], DLM is only meant to support multi-level modeling of domains (cf. section 4.2.2). DLM relaxes the "strict metamodeling" doctrine [2], to allow connections [13] to cross level-boundaries or orthogonal ontological classification dimensions. The latter are currently unique to DLM and support the separation of domain-induced *classification clusters* [19, p. 551].

Among further differences to Melanee are that tool support for DLM is in development only and that tool support will ideally support ontological instantiation by checking for semantic rather than literal conformance between instances and their types [9]. In contrast to Melanee, DLM does not consider both durability and mutability per feature, rather only the former in the form of "feature potency", in addition to "clabject potency" [17].

DLM is currently not associated with any particular constraint language. For the purposes of this paper, we are going to use DOCL as well. Further differences between Melanee and DLM will be discussed in section 4.

3 DESCRIPTION OF THE CHALLENGE SOLUTIONS

In this section, we describe the two solutions. Table 1 summarizes how the challenge requirements [25] were addressed by each solution. Note that the table only captures the summary points 1)–13) of the challenge and therefore does not cover the fact that the challenge description elsewhere restricts mobile phone factories to produce mobile phone devices only. Both of our solutions nevertheless implement the latter requirement.

3.1 Melanee Solution

Figure 1 shows the Melanee model with three levels (O_0–O_2). The top O_0 level contains *FactoryAsModelSupporter* with its specializations *MobilePhoneFactoryAsModelSupporter* plus *HuaweiMPFactoryAsModelSupporter*, and *CompanyAsOwner*, and *DeviceModel* with its specializations *MobilePhoneModel* plus *HuaweiMPModel*. With the exception of *DeviceModel* and its specializations, all clabjects have a potency value of "1", which specifies that instances of these clabjects will not be able to have instances of their own. Clabject *DeviceModel* and its specializations, have a potency value of "2" and therefore allow their instances in the middle level O_1 to have instances at the bottom O_2 level. The durability-2 attribute *RAM* specifies that every instance of *MobilePhoneModel* has to have a *RAM* attribute and hence that instances of the latter must have a *RAM* slot. The top three clabjects in level O_1, *Factory*, *Company* and *Device*, are not ontologically-typed (i.e. are not instances of a user-defined clabject). Clabjects *Factory* and *Device* are abstract (i.e. must not have any direct instances) which is represented by the fact that their potencies are "0". *Factory* is a superclass of *MP_Factory* and *HuaweiMP_Factory*, the latter being the only potent clabject in this inheritance hierarchy. *Device* is the superclass of *MP_Device*, which is an instance of *MobilePhoneModel*. *HuaweiMPDevice* and *MP_Device* are abstract clabjects, and the latter introduces an *IMEI* attribute with durability and mutability values of "1". The *produces* connection between *Factory* and *Device* is specialized via an inheritance relationship to reflect that only *MP_Factory* instances can produce devices of type *MP_Device*.

Clabjects *S400* and *S500* are device models (by virtue of being instances of *HuaweiMPModel* which in turn specializes *DeviceModel*) and can be instantiated to create objects that represent devices. Both device models are connected to an instance of *HuaweiMPFactoryAsModelSupporter*, named *Factory124*, and an instance of *CompanyAsOwner*, named *Huawei*. The O_1-level pair *Factory124* and *Huawei* have the same linguistic names (the ones appearing in the name compartment of clabject renderings) as the respective O_2-level instances of the O_1-level clabjects *HuaweiMP_Factory* and *Company*. These naming correspondences represent the fact that the matching elements model the same real-world objects respectively, but at different levels of abstraction. In other words, the *CompanyAsOwner* and *Company* instances represent the same real-world object called "Huawei", while the *HuaweiMPFactoryAsModelSupporter* and *HuaweiMP_Factory* instances represent the same real-world object called "Factory124". This dual-level representation of domain entities is designed to comprehensively cover the challenge specification in the presence of Melanee's "strict metamodelling" requirements that prohibit level-crossing connections.

Constraints. To ensure that the model in Figure 1 is well-formed with respect to the dual-level representations of "*Huawei*" and "*Factory124*", two constraints are needed: Constraints 1 & 2 ensure that any *CompanyAsOwner* instance being linked via *owns* to a *FactoryAsModelSupporter* instance has a name-corresponding pair of a *Company* instance being linked via *owns* to a *Factory* instance, and vice versa. The constraints are very similar in nature, the only difference is the direction for checking the existence-implication of the connections. Checking them in both directions ensures equivalence between the *owns* connections at level $O_1$and at level O_2.

Requirement	Melanee	DLM
1) A company has (a) a name, (b) owns factories, (c) owns device models	*Company* (O_1) has (a) a *name* attribute & (b) connects to *Factory* via *owns*. (c) *CompanyAsOwner* (O_0) is connected to *DeviceModel* via *owns*	*Company* (C_1) has (a) a *name* attribute and two *owns* associations, one to (b) *Factory* and one to (c) *DeviceModel*
2) Huawei is a (a) company that (b) owns Factory124 and (c) owns mobile phone models S400 and S500	*Huawei* is represented twice, once (a) as an instance of *Company* owning *Factory124*, and once (b) as an instance of *CompanyAsOwner* owning the *S400* and *S500* device models	(a) *Huawei* is an instance of *Company* (b) has an *owns* link to *Factory124*, and (c) has two *owns* links to the Huawei phone models *S400 & S500*
3) A factory (a) produces devices, (b) supports a list of device models, (c) can only produce devices that conform to (are of) supported device models	(a) *Factory* is connected to *Device* via *produces*, (b) *FactoryAsModelSupporter* is connected to *DeviceModel* via *supports*, (c) Constraint 5 ensures that a *Factory* only produces devices that are supported by the corresponding *FactoryAsModelSupporter*	(a) *Factory* (F_1) has a *produces* association to *Device*, and (b) has a *supports* association to *Device Model*. (c) Alignment of production of devices with supported models is realized by Constraint 7
4) A device conforms to a device model	Devices are instances of device models	Devices are instances of device models
5) A device model captures what is universal about the devices it describes	Device models describe devices by being the types of the latter	Device models are types for devices
6) A mobile phone model (a) allows specific RAM size options and (b) is a device model	(a) Constraint 4 specifies the *RAM* options for a mobile phone model, (b) *MobilePhoneModel* inherits from *DeviceModel*	(a) *Mobile Phone Model* defines *RAMoptions* which is populated by mobile phone models, (b) *Mobile Phone Model* specializes *Device Model*
7) A mobile phone device (a) conforms to a mobile phone model, (b) has an IMEI and (c) has a RAM size	(a) Mobile phone devices are instances of mobile phone models and the latter define (b) *IMEI* via inheriting from *MP_Device*, and (c) receive a *RAM* attribute from *MobilePhoneModel*	(a) Mobile phone devices are instances of mobile phone models which are in turn subtypes of *Mobile Phone Device*, the latter mandates an (b) IMEI number and a (c) RAM size via features
8) A mobile phone factory supports mobile phone models only	Association *supports* between *MobilePhoneFactoryAsModelSupporter* and *MobilePhoneModel* is a specialization of the more general *supports* association, which restricts the supporting devices.	Association specialization is used to restrict the support of *Mobile Phone Factory* instances to *Mobile Phone Model* instances
9) A Huawei mobile factory (a) supports Huawei mobile phone models only, (b) keeps track of mobile phone devices it produced, and (c) constrains the IMEI of the mobile phone devices produced by the factory to start with '001'	(a) In level O_0 the *supports* association is specialized to restrict support between *HuaweiMPFactoryAsModelSupporter* and *HuaweiMPModel* instances (b) *HuaweiMP_Factory* inherits a *produces* association from *MP_Factory* (c) Constraint 3 ensures that the IMEI starts with "001"	(a) Association specialization is used to restrict the support of *Huawei Mobile Phone Factory* instances to *Huawei Mobile Phone Model* instances, (b) *Huawei Mobile Phone Factory* has a respective *produces* association, (c) *Huawei Mobile Phone Factory* defines the prefix "001" which is used in *IMEI* in *Huawei Mobile Phone Device*
10) Factory124 (a) is a factory, (b) supports Huawei S400 and S500 mobile phone models, and (c) produced two S400 devices (S400_001, S400_002)	(a) *Factory124* at O_2 is an indirect instance of *Factory*, (b) at O_1, it *supports* the *S400* and *S500* mobile phone models, (c) and at O_2 it entertains *produces* links with *S400_001* and *S400_002*	(a) *Factory124* (F_0) is an indirect instance of *Factory*, (b) has *supports* links to *S400 & S500*, and (c) has *produces* links to *S400_001 & S400_002*
11) S400 (a) is a mobile phone model and (b) has either 4GB or 8GB of RAM	(a) *S400* is of type *HuaweiMPModel* which specializes *MobilePhoneModel*, (b) the RAM choices are enforced via Constraint 4	(a) *S400* is an indirect instance of *Mobile Phone Model* and (b) defines a *RAMoptions* list containing the 4GB and 8GB options
12) S400_001 (a) is a mobile phone device, (b) conforms to the S400 model, (c) has 4GBs of RAM, and (d) has '001468723648726' as its IMEI	*S400_001*, (a) is an indirect instance of *MP_Device*, (b) is typed by *S400*, (c) has 4GB of RAM, and (d) the IMEI attribute has the specified value	*S400_001* is (a) an indirect instance of *Mobile Phone Device*, (b) an instance of *S400*, (c) chooses *option 1* (4GB) of *RAMoptions*, and (d) specifies the IMEI suffix following "001" via its *id* value
13) S400_002 (a) is a mobile phone device, (b) conforms to the S400 model, (c) has 8GBs of RAM, and (d) has '0018768768475638' as its IMEI	Analogously to requirement 12), *S400_002* (a) is an indirect instance of *MP_Device* (b) is typed by *S400*, (c) has 8GB of RAM, and (d) the IMEI attribute has the specified value	*S400_002* is (a) an indirect instance of *Mobile Phone Device*, (b) an instance of *S400*, (c) chooses *option 2* (8GB) of *RAMoptions*, and (d) specifies the IMEI suffix following "001" via its *id* value

Table 1: Requirements of the challenge and their realization in Melanee and DLM

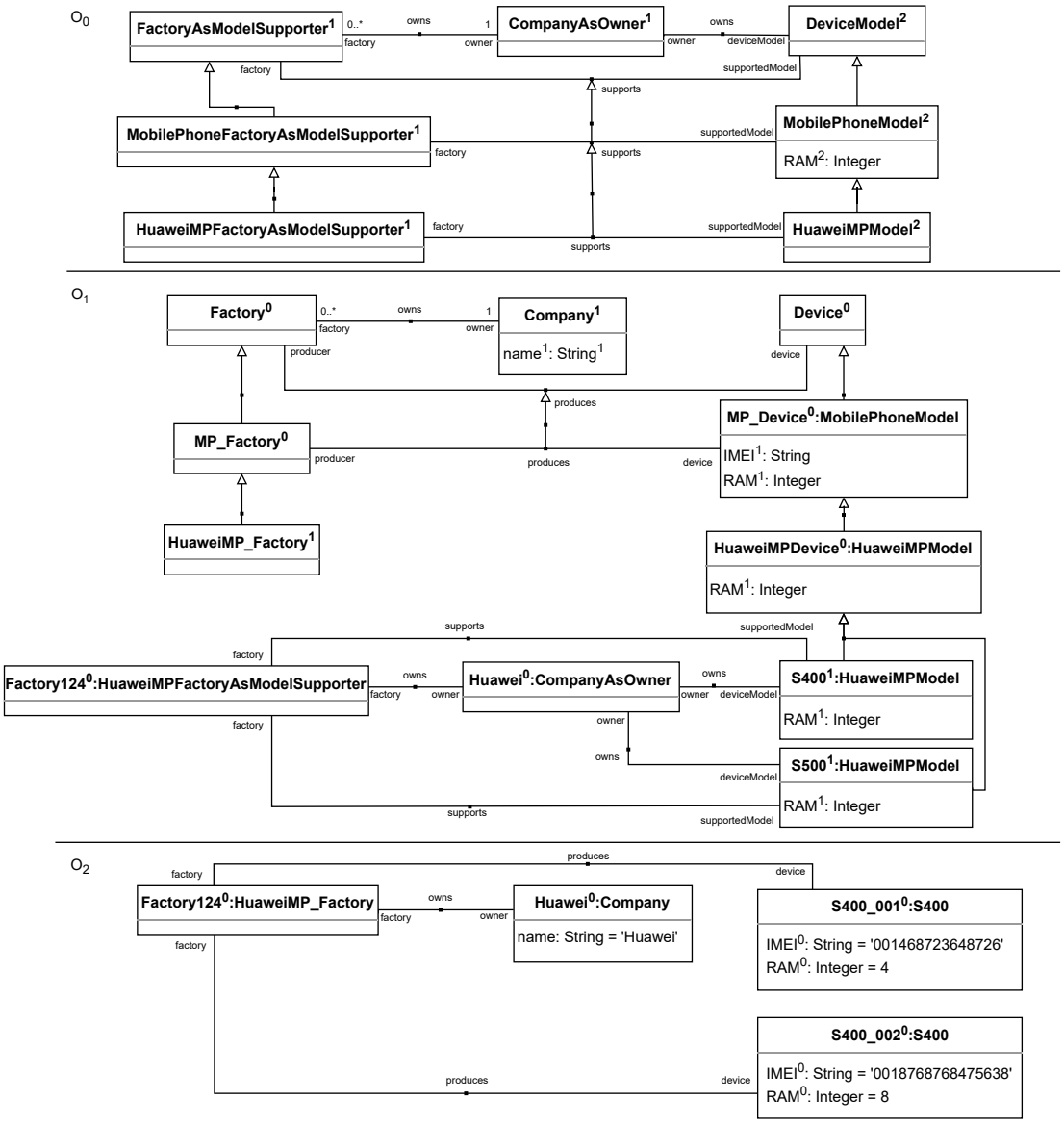

Figure 1: Melanee Solution of the collaborative challenge

```
context Company(2_2)
inv: let companyName = self.#name# in Clabject -> select(clabject|
     clabject.isDeepOclTypeOf(CompanyAsOwner)) -> select(
     companyAsOwner|companyAsOwner.#getPotency()# = 0) -> select(
     companyAsOwner|companyAsOwner.#name# = companyName).factory
     -> collect(#name#) -> includesAll(self.factory -> collect(#
     name#))
```

Constraint 1: Linking Company to CompanyAsOwner

```
context CompanyAsOwner(1_1)
inv: let companyAsOwnerName = self.#name# in Clabject -> select(
     clabject|clabject.isDeepOclTypeOf(Company)) -> select(company
     |company.#getPotency()# = 0) -> select(company|company.#name#
      = companyAsOwnerName).factory -> collect(#name#) ->
     includesAll(self.factory -> collect(#name#))
```

Constraint 2: Linking CompanyAsOwner to Company

The requirements state that every phone produced in a Huawei Mobile Phone Factory has to have an IMEI that starts with "001". This is guaranteed by Constraint 3.

```
context HuaweiMP_Factory(2_2)
inv: self.device -> forAll(IMEI.substring(1,3) = '001')
```

Constraint 3: IMEI prefix

It navigates from *HuaweiMP_Factory* instances to their produced devices and enforces that the first three digits of the IMEI are "001".

Constraint 4 limits the values of *RAM* slots of all *S400* instances to either "4" or "8".

```
context S400(2_2)
inv: self.RAM = 4 or self.RAM = 8
```

Constraint 4: Mobile Phone Model RAM option

Constraint 5 ensures that every factory only produces the devices that conform to device models that it supports. For every device produced by a certain O_2-level factory, its type must be among the models that are supported by the O_1-level representation (*FactoryAsModelSupporter* instance) of that factory (with which it shares the same name).

```
context Factory(2_2)
inv: let factoryName:String = self.#name# in
       let factoryTypeRole = Clabject -> select(clabject|clabject
     .isDeepOclTypeOf(FactoryAsModelSupporter) -> select(clabject|
     clabject.#getPotency()# = 0) -> select(clabject|clabject.#
     name# = factoryName) in
       self.device -> forAll(device | factoryTypeRole.
     supportedModel -> includes(device.#getDirectTypes()# -> first
     ()))
```

Constraint 5: Factory supported devices

Finally, Constraint 6 establishes that all instances of *HuaweiMPModel* must specialize *HuaweiMPDevice*, so that their instances are guaranteed to have the features specified in *HuaweiMPDevice*.

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

Constraint 6: Linking devices with models

Note that the solution described above differs from the 2021 "Melanee solution" [10]. We introduced more top-level concepts, e.g., *MobilePhoneModel* or *HuaweiMPFactoryAsModelSupporter* and respective connections, to more accurately satisfy the requirements regarding the specified refinement of "supports" connections.

3.2 DLM Solution

Since Melanee and DLM share many fundamental language design decisions, the models in Figure 1 and Figure 2 share many similarities regarding how domain elements are categorized, i.e., what their types and supertypes are. Both approaches furthermore use the same concept of connection specialization (cf. "association specialization" [27]) to restrict the domain and co-domain of elements in relationships between concepts in parallel specialization hierarchies. Therefore, many of the elaborations in section 3.1 carry over to the model in Figure 2. However, there are a number of notable differences.

The DLM model is structured in three main parts, so-called classification dimensions "C" (Company), "F" (Factory), and "P" (Product), each featuring its own level hierarchy (e.g., P_0–P_2 within in the "P" dimension). The underlying concept is that a domain scenario often contains so-called *classification clusters* – in the example, they are formed by company-, factory-, and product-concerns – which give rise to

- separable classification hierarchies, with
- individual hierarchy depths.

Unlike the product concern, which naturally gives rise to three levels of classification in both Melanee and the DLM solutions, the company and factory concerns do not require more than two levels.

DLM supports orthogonal ontological classification [19], i.e., allows classification dimensions to overlap in the sense that a single element can be classified simultaneously by more than one dimension. However, since the domain scenario of the challenge does not feature any overlapping classification, the solution uses multiple classification hierarchies for their organizational effect only.

Each of the three classification hierarchies "C", "F", and "P" enforces local well-formedness rules on the elements within a hierarchy but allows unrestricted inter-hierarchy connections to other hierarchies. In contrast, the Melanee model organizes all modeling elements within a single (O_2–O_0) hierarchy. This "single linear hierarchy" approach, combined with its adherence to the strict metamodeling doctrine, requires the Melanee model to use dual representations of *Factory124* and *Huawei* respectively. This kind of element duplication is absent in the DLM model, since for example the F_0 element *Factory124* in Figure 2 is allowed to maintain links to elements at level P_1 and P_0 across classification hierarchies. From the perspective of the F hierarchy, it does not matter in which specific P-level referenced elements reside. The respective *supports* and *produces* links certainly do not have to cross any F-level boundaries, even though they would have been allowed to in DLM, due to the latter's relaxation of the strict metamodeling doctrine.

Another difference between the solutions is how the IMEI and RAM size constraints are handled. Instead of checking the respective values with constraints (cf. Constraints 3 & 4), the DLM solution ensures they are correct by construction, thus forgoing the need for any respective textual constraints. P_1-level element *Huawei Mobile Phone Device* redefines the *IMEI* signature declaration with an operation *IMEI* that constructs the IMEI value for Huawei mobile phone devices by prefixing their (IMEI-) ids with a value ("001") obtained from F_1-level element *Huawei Mobile Phone Factory*.

In a similar fashion, a mobile phone device specifies its RAM size via selecting one of the valid options available from its corresponding mobile phone model. In Figure 2, the mobile phone model *S400* makes the options "4GB" and "8GB" available. The actual RAM size of a mobile phone device (e.g., *S400_001*) is then determined by evaluating operation *RAM*, defined in *Huawei Mobile Phone Device*, which references the RAM option value of *S400_001* (*option = 1*) and uses it to index the available RAM options defined in *S400*.

A number of notational differences exist between the models, for example in the DLM model, element names are underlined if they represent individuals in the domain, i.e., cannot be instantiated any further. Second, potency values are only explicitly specified if they are needed to restrict the effect depth of a clabject or feature. This is the case for *RAMoptions* at level P_2 which would otherwise be interpreted as a deep field whose value assignments would occur at level P_0. Since all fields in the DLM model are so-called "single fields" (as opposed to "dual fields") [6], the use of "=" for value assignments implies a potency-zero value for the field, i.e., no explicit potency value has to be provided. Third, the DLM model uses explicit visual *instance of* relationships to connect ontologically-typed elements to their types. It would have been possible to use the same "*elementName : elementType*" notation as used in the Melanee model but the idea was to

- emphasize the three different classification hierarchies, and
- visually highlight ontological classifications.

The DLM solution does not restrict any clabjects to be abstract classifiers since the challenge requirements do not make such stipulations. If desired, the same potency-zero approach can be used in DLM (cf. *Device* in Figure 1).

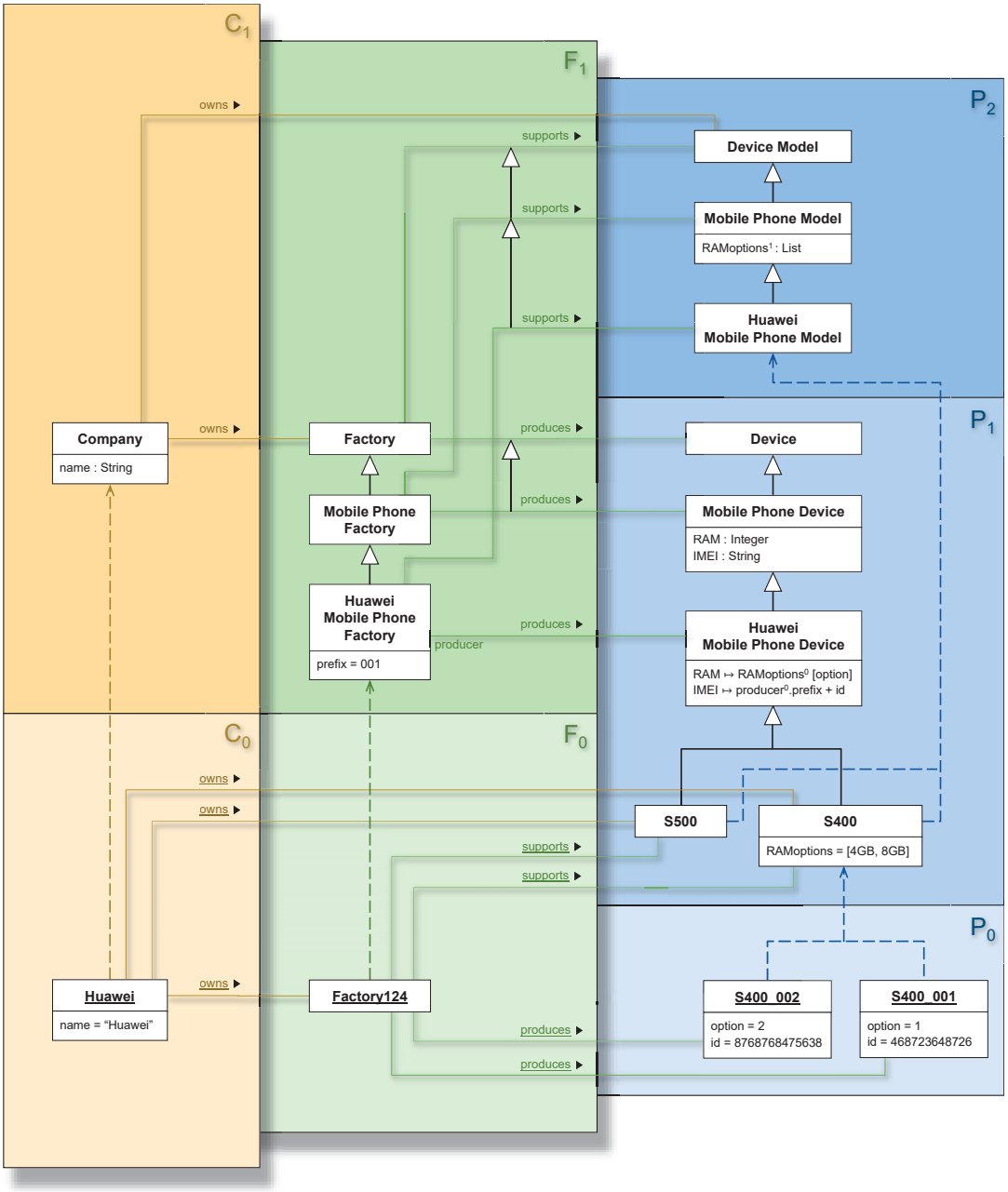

Figure 2: DLM Solution of the collaborative challenge

Constraints. Only two textual constraints need to accompany the model of Figure 2. The first one (Constraint 7) addresses requirement 3) in Table 1, i.e., ensures that every factory only produces the devices that conform to device models that it supports.

```
context Factory
inv: self.produces -> forAll(device |
  self.supports -> includes(device.#getDirectTypes()#->first()))
```

Constraint 7: Factory supported devices

The second constraint (Constraint 8) ensures that *Huawei Mobile Phone Model* instances are subtypes of *Huawei Mobile Phone Device*. If that were not enforced then the second-order instances of *Huawei Mobile Phone Model*, i.e., actual Huawei mobile phone devices, would not have to conform to the stipulations made in the specialization hierarchy that has *Huawei Mobile Phone Device* at its bottom.

```
context Huawei Mobile Phone Model
inv: self.#getSuperTypes()# -> collect(#name#)
       -> includes("Huawei Mobile Phone Device")
```

Constraint 8: Linking devices with models

4 DISCUSSION

In this section we compare the two solutions, highlighting similarities and differences of both the solutions and approaches, and investigate the respective trade-offs.

4.1 Similarities

Unsurprisingly, due to their shared origin, the Melanee and DLM solutions share a number of similarities.

4.1.1 Nature of Levels. Since levels in both Melanee and DLM are classification levels and use inter-level classification relationships that mirror logical classification in the domain, the concepts used in the solutions and their placement in the levels hierarchy are in essence identical, modulo the use of classification dimensions for the DLM solution and the partial element duplication in the Melanee solution. Both solutions

- distinguish between classification and generalization,
- constrain generalization relationships within a level, and
- restrict level-crossing relationships within a level hierarchy to the *instance of* relationship only.

Due to the fact that both solutions require textual constraints (cf. Constraints 6 & 8) to ensure consistent modeling, and the respective requirement to link instances of metatypes to a particular supertype is not uncommon, it could be worth considering for both approaches to adopt a model-integrated alternative to textual constraints, such as the level-crossing *categorizes* relationship (a weaker form of the *isPowertypeOf* relationship) of MLT [12]. Such explicit relationships are recognizable in a model without having to consult constraints and do not require the modeler to be familiar with a textual constraint notation. Furthermore, model creators would be shielded from accidentally getting a detail of the powertype constraint wrong, and model maintainers would not have to ensure that element names occurring in powertype constraints are changed if the corresponding model element names are changed.

4.1.2 Conformance Modeling. Both Melanee and DLM solutions directly map a number of relationship concepts used in the requirements, including "*X conforms to Y*", "*Y describes X*", "*X is of Y*", and "*X is a Y*", to their built-in, inter-level "*X instanceOf Y*" relationship. This was possible because in each of the above domain relationships, *X* is an individual that in some shape or form needs to conform to a universal *Y* and none of the domain relationships require conformance checking that goes beyond what classification in Melanee and DLM can model.

However, in general, one cannot assume that all similar domain notions will semantically match with a built-in modeling relationship, e.g., when a looser notion of conformance than literal conformance between model elements is required. Thus, relying on a built-in relationship like "*instanceOf*" to model a "*conformance*" requirement and reaping the benefits of concise models and leveraging built-in conformance checking, is only possible if conformance checking is not compromised by mapping it to "*instanceOf*"-based conformance checking.

4.1.3 Deep Characterization. Melanee and DLM can employ two forms of deep characterization, i.e., the ability to let an element affect another element located more than one level below it, which are a "powertype"-style and a potency-based form of deep characterization. Both solutions embody the former for IMEI attribute and the Melanee solution embodies the latter for the RAM attribute. The *MobilePhoneModel* entity introduced the RAM attribute with a durability value of "2", which means the attribute will be present in second-order instances of that clabject, i.e., at the bottom O_2 level.

We interpreted the challenge description as not only requiring a necessary scaffolding to support the described bottom-level objects, but also as strongly suggesting important concepts, such as "mobile phone device" and "mobile phone model" as desirable explicit anchors for future extensions to the scenario, i.e., supporting the addition of more phone companies, their factories, and produced devices. As this resulted in having to link instances in the middle-level to supertypes in the middle level, the powertype constraints (Constraints 6 & 8) would have been required anyhow. Hence, we implemented the IMEI feature with the "powertype"-style.

4.1.4 Modeling Notation. Both approaches use a visual notation that is closely aligned with the UML but is level-agnostic, i.e., avoids notational differences based on level membership ("class level" vs "object level" in the UML) [5].

There are minor differences such as the indication of a collapsed association class through a small dot in the middle of connections in Melanee (see Figure 1), even when no association class is present, but overall there are no fundamental notational differences. A major difference, however, is that Melanee comes with a full-fledged custom-visualization approach that allows users to define custom-domain-specific renderings of their models [16].

4.1.5 Constraints. Since DLM is not a full-fledged product like Melanee yet, there is currently no constraint language associated with it. However, an OCL-like constraint language aligned with the OCA, such as DOCL, is a very good fit for DLM.

The solutions do not use the same approach to enforce the *IMEI* and *RAM* requirements (see Table 1), but the differences are of a stylistic nature as opposed to being caused by technological differences. The approach taken in the DLM solution is to replace straightforward object data slots by operations that yield the same value, based on supporting values provided by the objects.

Taking *IMEI* as an example, this results in

- values that are correct by construction,
- avoiding the redundant representation of the mandatory "001"-prefix in all Huawei mobile phone devices,
- associating the "001" value with the domain concept *Huawei Mobile Phone Factory* that stipulates it, and
- making the "001" prefix explicitly visible and editable in the model (see Figure 2).

The alternative to directly associating values such as the *IMEI*-prefix or the available *RAM* sizes with the model elements that represent the domain concepts that stipulated these values, is to use them inside of constraints (cf. Constraints 3 & Constraint 4).

The downside of the operation-based approach is that the final values are not visible at the object level, unless the environment performs on-the-fly evaluations of operations and displays the operations and their results along with the objects.

4.2 Differences

4.2.1 Strictness. Both Melanee and DLM support sanity-checking of models by running well-formedness checks on models. They thus can "safeguard against ill-formed models" [18], including those containing anti-patterns [11, 14]. The respective ability to detect and reject models that lack a sound set-theoretic interpretation is akin to static type checking in programming languages, i.e., it can catch issues early that otherwise may cause problems much later in development.

Melanee's enforcement of "strict metamodeling" [2] provides the above benefits but also sometimes requires modelers to use workarounds to circumnavigate the strictness requirements. In this case the fact that *supports* connections from a sole *Factory124* object at the bottom level to *S400*/*S500* are illegal in a Melanee model, required the already mentioned dual-level representation workaround. The latter not only resulted in some amount of element duplication – *Factory124* & *Huawei* needed to be represented twice and *FactoryAsModelSupporter* & *CompanyAsOwner* were required in addition to *Factory* and *Company* – but also necessitated two additional constraints (Constraints 1 & 2) to ensure that the dual-level representation is maintained consistently. Finally, another constraint (Constraint 5) required a more involved formulation than a functionally equivalent constraint (cf. Constraint 7) because it had to navigate the dual-level representation.

In contrast, due to DLM's sound relaxation of the "strict metamodeling" doctrine [19], the DLM solution does not require any workarounds.

4.2.2 Accidental Complexity. In [20], Atkinson and Kühne refer to "*accidental complexity*" as being caused by "*. . . mismatches between a problem and the technology used to represent the problem.*" They specifically mention workarounds as a source of accidental complexity, referring to workarounds "*. . . being used because an isomorphic relationship between domain concepts and modeling language concepts is not possible*". In this sense, the Melanee solution suffers from accidental complexity that cannot be observed in the DLM solution (see section 4.2.1). Melanee accepts this downside to enable the strong sanity-checking qualities of "strict metamodeling". DLM's slight relaxation of the latter doctrine could theoretically weaken sanity-checking support but at the current stage of investigation, no such loss in sanity-checking support is known.

A trade-off of DLM's relaxation of the "strict metamodeling" doctrine is the fact that by allowing connections to cross level boundaries, it is no longer possible to regard the contents of a single level as a complete language definition (for the level below) or a pure language usage (of the language definition above). Confining language definition and/or usage to a single level might be regarded as achieving a form of enhanced "vertical modularity". Due to the frequent occurrence of "linguistic extensions" [15], i.e., elements without an ontological type, this property does not literally hold in Melanee either, but could in principle be established by augmenting O-levels with the necessary linguistic concepts.

However, if such level-confined language definitions/usages are desirable for some reason, it would be possible to optionally enforce that connection ends are always located within the same level with an optional DLM limitation.

4.2.3 Ontological Dimensions. As mentioned before, DLM supports multiple classification hierarchies within one model, allowing elements in those hierarchies to be connected to each other across hierarchies. Provided respective visual presentation support is used for highlighting hierarchies within a model, it is possible to make it easier for a modeler to recognize any modular structures within an overall model (cf. the three hierarchies in Figure 2), i.e., appreciate a "horizontal structure" implied by domain concerns in addition to the "vertical structure" implied by domain classification.

Note, however, that the colors or the backgrounds in Figure 2 purely serve a presentational purpose. A modeler does not have to use these presentational aids to be able to sanity-check their model. The classification hierarchies can be inferred from the *instance of* relationships in the model. Some modelers may find it difficult to understand feedback from a DLM tool when it reports some well-formedness issue and could take some time to figure out which part of the model would have to change to make it sound. In contrast, a Melanee user would not encounter such situations as Melanee would not allow the modeler to construct an unsound model in the first place. A usability experiment would be required to settle the question of whether it is better to tie the modeler's hands so that they cannot create issues (but must use workarounds such as dual-level representations), or let the modeler proceed and provide as helpful as possible well-formedness violation reports, potentially accompanied by "quick fix" menus. The premise underlying DLM is that modelers will more often than not intuitively create a model like the one in Figure 2 and not encounter any error messages. In other words, the expectation is that user-created models will generally be sound, despite the presence of multiple domain concerns and hierarchy- or level-crossing relationships, and that the benefits of sanity-checking will only sometimes kick in, when modelers are confused about some concepts.

4.2.4 Language Complexity. DLM's language design is more complex than Melanee's underlying language design. Abilities such as overlapping classification and the associated multiple potencies per element (one per overlapping dimension the element is involved in), require additional notational support and an adequate understanding of the underlying concepts on behalf of the modeler to be applied correctly. To fully master DLM and apply it to optimal effect thus requires more language competence in comparison.

While less complex modeling languages are to be preferred, everything else being equal, it is often the case that simpler languages burden models with additional complexity in the form of workarounds (cf. section 4.2.1) or realization effort, i.e., the use of many primitive elements in a solution to combine to the effect of one complex concept (cf. RISC vs CISC in chip design). The downsides of a more complex modeling language design therefore have to be weighed against the downsides of the effects less complex languages have on model complexity. While additional concepts in a richer language need to be mastered, so do the workaround and realization techniques necessary for a simpler language which will furthermore imprint additional complexity on the model.

4.2.5 Potency. There are several differences regarding the support of "potency" between Melaneed and DLM.

Melanee's "*star*-potency" (*) allows the potency of a clabject to be specified as supporting arbitrary instantiation depths, akin to a

"wildcard" specification [16]. In contrast, DLM regards the explicit specification of a clabject potency as constraining its maximum instantiation depth and therefore interprets the lack of any such explicit value as "underspecification", i.e., the absence of any such limitation (cf. the definition of *characterization potency* [17]).

Melanee supports two so-called "vitality" properties per field (attribute or slot): "duration" and "mutability" (see section 2.1). In comparison, DLM only supports field durability, referring to it as "feature potency" and currently has no mechanism to achieve field immutability after a specified number of instantiation depths. The DLM philosophy is to avoid duplicating the same constant value downwards an instantiation chain and rather encourage access of a shared value (which might be constant or not) through upward type inquiries, similar to how "static" (i.e., class-level) values can be shared among instances in programming languages.

DLM supports both "single-" and "dual-"fields [6], i.e., fields that may only have a value if their potency is zero, or at any location in an instantiation chain, respectively. Melanee only uses the latter kind, i.e., all fields may have values at any instantiation depth. This makes the Melanee design simpler and easier to learn, but forces the existence of values even when they have no ontological justification at a certain level. While it is often possible to assign some useful interpretation to a value, for example at a type level when the primary values are at the object level, e.g., as a "default value", DLM maintains that there is merit in using single fields to signify that no reasonable ontological interpretation of values along the instantiation path exists for some fields.

4.2.6 Level Numbering. In both approaches, the bottom level(s) in the classification hierarchies hosts objects, i.e., clabjects that represent individuals from the domain and cannot be further instantiated.

In DLM a bottom level is indexed with "0" and the levels containing type abstractions higher up receive increasingly higher indices, e.g., P_1, P_2, etc. DLM considers the logical level of individuals in the domain as the foundational one and maintains there is no natural limit to the type of abstractions one can form over them, thus placing the "open" end of hierarchies at the top. This level numbering scheme favors a bottom-up modeling approach, in which objects are identified first, then their types, then the types of the latter, etc.

In Melanee, the topmost level receives the index "0", e.g., O_0 as in Figure 1, with levels further down receiving increasingly higher indices, e.g. O_1, O_2, etc. This favors a top-down modeling approach in which general, reusable abstractions are identified first and then the level hierarchy is allowed to grow to become as deep as needed, typically to reach elements that represent individuals in the domain, or at least down to the level containing the types of the latter, in case modeling objects is not important/helpful.

Whether level numbers ascend or descend does not matter for finalized hierarchies since only the total order property of the indices is required for well-formedness checking. The direction of the numbering scheme makes a difference, however, if the final height of a hierarchy is unknown when modeling commences. A DLM modeler would be ill-advised to arbitrarily start populating higher levels, e.g., P_2 first, with what they presume to be metaypes, in this example. If it should turn out that they have misjudged where objects should end up, they would have to renumber levels accordingly. As such, a DLM modeler should typically start with modeling individuals, to then add abstractions on top of them.

Conversely, it can be problematic for a Melanee modeler to place an element representing a domain individual at some deeper level O_n first, only then to find out that they have misjudged how many classification levels will be needed overall. In order to create the best chances for avoiding any renumbering, a Melanee modeler needs to first identify the most abstract elements, place them at O_0, and then work downwards in a step-wise fashion.

It is worth noting that level renumbering could be automated, i.e., need not necessarily imply manual effort on behalf of the user, however, renumbering occurrences would still be a burden to modelers, if the latter associate certain concepts with certain level numbers. It therefore still matters under which circumstances the numbering has to be updated.

Note that while adding a further classification level to the top of a Melanee hierarchy necessitates level renumbering, the latter becomes necessary if a level is added to the bottom of a DLM hierarchy. Such a bottom-level addition would imply, though, that the elements previously thought to represent individuals now represent classes of individuals, thus requiring a re-conceptualization of these elements, as well as of those elements that are above them. Which of the two approaches is more robust with respect to renumbering therefore depends on whether adding a further level of abstraction or a complete re-conceptualization of the entire hierarchy is expected to be more likely.

The renumbering issue is somewhat alleviated by Melanee's "*star*-potency" which modelers may use to avoid associating a predefined instantiation depth with a top level, also allowing the latter to be reused in multiple contexts (see section 4.2.5). To be able to benefit from potency specifications, though, concrete numbers have to be introduced as some point and then form a commitment to the remaining instantiation depth similar to choosing concrete numbers at the top.

In principle, it is possible to use level names and absolute level references [26] to avoid any renumbering issues. However, a level numbering scheme that does not imply a total order is not applicable to classification-based approaches like Melanee and DLM. The latter do not support bottom-extensions beyond elements that represent individuals in the domain or arbitrary level insertions anyhow and hence could not make good use of unordered indices.

4.2.7 Maturity. Melanee has a working EMF-based implementation, which has been tried and tested and comes with numerous features such as an emendation service, constraint evaluation, transformations, custom-visualization, etc. [3, 16], and is available to download [24]. The tool supports modeling in graphical, text-based, table-based and/or form-based formats to provide various stakeholders with multiple ways to interact with models. Melanee has been put to practice in research and education for years, thereby building trust in its multi-level modeling language design choices.

In contrast, DLM so far is only supported by a prototype environment and therefore predominantly exists as a conceptual language design. Although it shares a long heritage with Melanee in the form of a common past, in as much it deviates from Melanee's underlying language design (see in particular section 4.2), it has not seen exposure to practice yet.

5 CONCLUSION

We presented two solutions to the MULTI *Collaborative Comparison Challenge* modeled using the multi-level modeling approaches Melanee and DLM. Both, although based on traditional object-oriented concepts, support modeling beyond the capabilities of UML class- and object diagrams while retaining many characteristics of the latter to ease the transition from two-level to multi-level modeling.

The presented models demonstrate that both approaches are capable of accurately modeling all challenge requirements. However, their respective solutions differ in important ways.

The Melanee-based solution introduces accidental complexity to the model by using a dual-level representation workaround which is needed to cope with restrictions established by Melanee's adherence to the "strict metamodeling" doctrine. In general, whenever domain concepts entertain relationships to other concepts at different classification levels, such as *Factory* and *Company* in the challenge, when using Melanee it not only becomes necessary to explicitly include duplicate model elements in the model – each element representing the same real world object in a different role – it also becomes necessary to include additional constraints to ensure that the duplicates are used in a consistent manner. While an individual representation of roles can sometimes aid clarity, it can also give rise to undesirable complexity in the form of model size increase, more difficult to follow relationships, and more involved element navigation in constraints. A number of approaches of adapting "strict metamodeling" to address this issue have been suggested, including (a) the "modeling spaces" approach where a high-order organization principle is used to separate non-aligning concepts into separate modeling spaces [7], or (b) a deep view-based approach in which strict views are projected from a more loosely modeled SUM [23, 28, Section 8].

DLM avoids the aforementioned introduction of accidental complexity by allowing connections to span across multiple classification hierarchies, or cross levels boundaries within single hierarchies. The respective relaxation of the "strict metamodeling" doctrine only removes the requirement that connection ends must be in the same level, it does not forgo well-formedness checking on connections in general. As a special kind of clabjects, connections are still subject to the same soundness establishing well-formedness rules that govern clabject relationships. Within each classification cluster, which includes respective implicit clabject- and connections-clusters, strict well-formedness rules apply, thus preventing a loss of sanity-checking ability, compared to other forms of loosening the constraints of strict metamodeling [19].

The question of whether Melanee's or DLM's approach results in better usability is a highly interesting one, in particular, if Melanee is amended with an alternative approach to representing level-crossing connections in domains. Language complexity, model complexity, intuitiveness, proximity to familiar approaches, are all aspects worthy of investigation in future evaluations.

We hope the solutions and discussion provided in this paper will help highlight the impact of some of the different design choices made in the two multi-level modeling approaches studied, thereby suggesting which features may be most beneficially passed on to future generations of multi-level modeling languages, and which features should be considered worth revising.

REFERENCES

[1] João Paulo A. Almeida, Thomas Kühne, and Marco Montali. 2022. Editorial to the Special Issue: Multi-Level Modeling Process Challenge. *International Journal of Conceptual Modeling* 17 (June 2022), 1–5. DOI: https://doi.org/10.18417/emisa.17.4

[2] Colin Atkinson. 1997. Meta-Modeling for Distributed Object Environments. In *Enterprise Distributed Object Computing*. IEEE, 90–101.

[3] Colin Atkinson and Ralph Gerbig. 2016. Flexible deep modeling with Melanee, In Modellierung 2016 - Workshopband. *GI-Edition / Proceedings* 255 (2016), 117–121. http://ub-madoc.bib.uni-mannheim.de/40981/

[4] Colin Atkinson, Ralph Gerbig, and Thomas Kühne. 2014. Comparing multi-level modeling approaches. In *CEUR Workshop Proceedings*, Vol. 1286. CEUR.

[5] Colin Atkinson and Thomas Kühne. 2000. Meta-Level Independent Modeling. In *International Workshop Model Engineering (in Conjunction with ECOOP'2000)*. Springer, Cannes, France.

[6] Colin Atkinson and Thomas Kühne. 2001. The Essence of Multilevel Metamodeling. In *UML 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, Martin Gogolla and Cris Kobryn (Eds.). Springer, 19–33.

[7] 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. DOI: https://doi.org/10.1142/S0218194001000724

[8] Colin Atkinson and Thomas Kühne. 2003. Model-Driven Development: A Metamodeling Foundation. *IEEE Software* 20, 5 (Sept. 2003), 36–41.

[9] Colin Atkinson and Thomas Kühne. 2016. Demystifying Ontological Classification in Language Engineering. In *Modelling Foundations and Applications*, Vol. LNCS 9764. Springer, 83–100.

[10] Sándor Bácsi, Arne Lange, Thomas Kühne, Gergely Mezei, and Colin Atkinson. 2021. Melanee and DMLA – A Contribution to the MULTI 2021 Collaborative Comparison Challenge. In *Proceedings of MULTI 2021*. IEEE, 556–565.

[11] 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 WWW'16*. ACM, 975–980.

[12] Victorio A. Carvalho and João Paulo A. Almeida. 2018. Toward a well-founded theory for multi-level conceptual modeling. *Software & Systems Modeling* 17, 1 (2018), 205–231. DOI: https://doi.org/10.1007/s10270-016-0538-9

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

[14] Atílio A. Dadalto, João Paulo A. Almeida, Claudenir M. Fonseca, and Giancarlo Guizzardi. 2021. Type or individual? Evidence of large-scale conceptual disarray in wikidata. In *International Conference on Conceptual Modeling*. Springer.

[15] Juan de Lara and Esther Guerra. 2010. Deep Meta-modelling with MetaDepth. In *Objects, Models, Components, Patterns*. Springer, 1–20.

[16] Ralph Gerbig. 2017. *Deep, seamless, multi-format, multi-notation definition and use of domain-specific languages*. Ph.D. Dissertation. University of Mannheim, München. https://madoc.bib.uni-mannheim.de/42010/

[17] Thomas Kühne. 2018. Exploring potency. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2–12. DOI: https://doi.org/10.1145/3239372.3239411

[18] Thomas Kühne. 2018. A story of levels. In *Proceedings of MODELS 2018 Workshops, Copenhagen, Denmark, 2018*. CEUR, http://ceur-ws.org/Vol-2245/, 673–682.

[19] Thomas Kühne. 2022. Multi-dimensional multi-level modeling. *Software and Systems Modeling* 21, 2 (2022), 543–559. DOI: https://doi.org/10.1007/s10270-021-00951-5

[20] Thomas Kühne and Colin Atkinson. 2008. Reducing accidental complexity in domain models. *Software & Systems Modeling* 7, 3 (01 Jul 2008), 345–359. DOI: https://doi.org/10.1007/s10270-007-0061-0

[21] Arne Lange. 2016. *dACL: the deep constraint and action language for static and dynamic semantic definition in Melanee*. Master's thesis. University of Mannheim. http://ub-madoc.bib.uni-mannheim.de/43490/

[22] Arne Lange and Colin Atkinson. 2019. On the Rules for Inheritance in LML. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 113–118.

[23] Arne Lange and Colin Atkinson. 2022. Multi-level modeling with LML. *International Journal of Conceptual Modeling* 17 (June 2022), 1–36. DOI: https://doi.org/10.18417/emisa.17.6

[24] Melanee. 2015. Melanee 2.0 Home Page. http://www.melanee.org. (2015).

[25] Gergely Mezei, Thomas Kühne, Victorio Carvalho, and Bernd Neumayr. 2021. The MULTI Collaborative Comparison Challenge. MULTI 2021 Call for Papers. (2021). https://jku-win-dke.github.io/MULTI2022/MULTI2021_Challenge.pdf

[26] Bernd Neumayr, Katharina Grün, and Michael Schrefl. 2009. Multi-level Domain Modeling with M-objects and M-relationships. In *Proceedings of APCCM'09*. Australian Computer Society, 107–116.

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

[28] Christian Tunjic. 2021. *A Deep Orthographic Modelling Environment*. Ph.D. Dissertation. University of Mannheim.