Postprint of the DKE article 102481 at https://doi.org/10.1016/j.datak.2025.102481

Supporting Sound Multi-Level Modeling

Specification and Implementation of a Multi-Dimensional Modeling Approach

Thomas Kühne [c,*], Manfred A. Jeusfeld [d]

[c] *Victoria University of Wellington, Wellington, New Zealand*
[d] *University of Skövde, Skövde, Sweden*

Abstract

Multiple levels of classification naturally occur in many domains. Several multi-level modeling approaches account for this, and a subset of them attempt to provide their users with sanity-checking mechanisms in order to guard them against conceptually ill-formed models. Historically, the respective multi-level well-formedness schemes have either been overly restrictive or too lax. Orthogonal Ontological Classification has been proposed as a foundation for sound multi-level modeling that combines the selectivity of strict schemes with the flexibility afforded by laxer schemes. In this article, we present the second iteration of a formalization of Orthogonal Ontological Classification, which we empirically validated to demonstrate some of its hitherto only postulated claims using an implementation in ConceptBase. We discuss the expressiveness of the formal language used, ConceptBase's evaluation efficiency, and the usability of our realization based on a digital twin example model.

Keywords: conceptual modeling, multi-level modeling, well-formedness, integrity constraints, modeling anti-patterns

1. Introduction

Modeling languages for conceptual modeling differ by the level of support they provide for modeling domains with multiple classification levels; specifically how explicitly they represent deep domain classification within models. A long history of modeling mechanisms that attempt to support the modeling of multiple classification domain levels includes materialization [1] and power-types [2], both implying concepts that go beyond individuals and their types. Telos [3] pioneered support for an unbounded number of classification levels and DeepTelos [4] added support for *deep characterization* [5].

*Corresponding author
Email addresses: tk@ecs.vuw.ac.nz (Thomas Kühne), manfred.jeusfeld@acm.org (Manfred A. Jeusfeld)

Unfortunately, having to manage more than two classification levels increases the potential of creating ill-formed models, i.e., models that cannot be given a sound interpretation. It has been argued that the complexity of contemporary conceptual modeling is akin to the complexity of programming large computer systems and therefore analog complexity management strategies are needed [6].

A well-known discipline within the area of multi-level modeling [7] for enforcing sound models is "strict metamodeling" [8], which is widely accepted to be highly selective, i.e., highly discriminative against ill-formed models, but has been equally widely criticized for being too inflexible [9, 10, 11].

Multi-dimensional multi-level modeling (MdM), based on the notion of "Orthogonal Ontological Classification" [12], has been proposed as a multi-level modeling paradigm that claims to enjoy the same selectivity as "strict metamodeling" but without incurring the latter's downside of requiring modelers to employ workarounds for several commonly occurring modeling scenarios. However, the initial description of MdM in [12] was informal only, i.e., did not include verification or validation of its claims.

As ConceptBase had been successfully used to formalize the multi-level modeling approaches DDI [13], DeepTelos [4], and MLT* [14], we set out to

- develop a formalization of MdM,
- investigate whether ConceptBase's deductive specification language is sufficiently expressive to support this formalization,
- examine ConceptBase's efficiency when supporting MdM, and
- empirically validate some of the MdM claims.

In this article, we first further motivate the need for well-formedness checking of models featuring multiple levels of domain classification and then briefly compare two existing approaches to MdM [12] in Section 2. We subsequently present an MdM formalization using many-sorted first-order logic in Section 3 and follow with a description of an implementation of the formalization using ConceptBase in Section 4. Section 5 presents a digital twin scenario, on which we base the performance analysis of our implementation in Section 6, comparing three alternatives to check well-formedness using ConceptBase. We precede our conclusions with a discussion of our formalization and its implementation in Section 7.

This article is an extended version of "Sanity-Checking Multiple Levels of Classification - A Formal Approach with a ConceptBase Implementation" [15]. It addresses the then open question of whether ConceptBase is able to realize the checking of the "*level-respecting*" property [16, 12, 15], presents a formalization update that considerably extends the range of models that can be rejected as ill-formed, contains an improved discussion on constraints vs queries, better documents how our implementation is embedded within ConceptBase, adds an example model that demonstrates the ConceptBase implementation, and reports on the evaluation efficiency of our implementation.

2. Sanity Checking

Enforcing well-formedness requirements on models or programs is a well-established technique to ensure that the latter have a sound semantics. In particular, well-formedness requirements have been effectively used as preconditions to the analysis, interpretation, execution, etc., of models, preventing semantics implementations from tripping over problematic structures such as circular definitions, dangling references, etc.

Beyond serving this purpose, however, well-formedness constraints may also be used to alert users to structures that would not necessarily create problems for semantics implementations, but instead represent conceptual issues such as performing a category mistake. Such structures are syntactically valid but logically flawed or inconsistent and are typically the result of ostensibly sound local contributions that unintentionally add up to an unsound model. In particular domains with multiple classification levels are more challenging with respect to avoiding conceptual issues in user models. In principle, such conceptual issues are harder to identify than violations of straightforward structural requirements since they technically involve the semantics of concepts.

Providing respective solutions is becoming increasingly important due to the dependence of societies on reliable data and the significant amount of higher-order concepts naturally arising not only in specialized domains such as biology, or process metamodeling, but also in such commonplace domains as covered by UNICLASS classifications [17] and Wikidata [18, 19]. Brasileiro et al. reported that in 2016 Wikidata contained 6,963,059 elements involved in instantiation chains of lengths three [18], testifying to the presence of domains which require more than two conceptual classification levels.

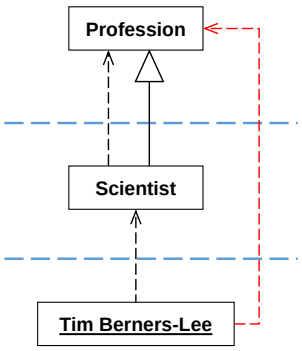


Figure 1: Semantically Flawed Model

2.1. Detecting Ill-Conceived Conceptualizations

Consider Figure 1 which, using UML class/object diagram notation [20], shows a condensed version of a modeling scenario that was part of Wikidata in 2016 [18, Figs. 3 & 4]. We use UML's dependency relationship to denote "instanceOf", e.g., Tim Berners-Lee is declared to be an instance of Scientist.

The rightmost "instance-of" relationship can be derived from two Wikidata claims: First, that Tim Berners-Lee is a scientist and, second, that "Scientist" is a subtype of "Profession". From these claims one can conclude that Tim Berners-Lee is a profession, which obviously does not make sense, hence the different coloring of this implied "instanceOf" relationship. Dadalto et al. observed that Wikidata no longer supports this particular incorrect inference, but that this is not a result of applying a general solution to eradicate all such issues [19]. Equivalently ill-formed model fragments, e.g., a certain "Frank Hilker" being inferable as a "Position" are still pervasive in Wikidata, affecting many areas including biology, gastronomy, awards, professions, and sports [19]. In order to detect such issues in Wikidata, Brasileiro et al. presented three "Anti-Patterns" that can identify ill-formed model fragments, such as the one in Figure 1. "Anti-Patterns" can be thought of embodying a modeling inconsistency in the form of a graph pattern that, if they match a model fragment, identify the latter to be ill-formed. Brasileiro et al. found that 15,177 Wikidata elements were involved in "Anti-Pattern 1", and 7,082 were involved in "Anti-Pattern 3" [18].

In general, such nonsensical model fragments cannot be mechanically detected without attaching semantics to the concepts involved and, based on those semantics, computing that a claim is made involving incompatible concepts.

Fortunately, however, nonsensical models like that in Figure 1 can still be mechanically detected without having to attach rich semantics to the concepts involved. For instance, by associating "order"-values to the concepts, e.g., by categorizing Tim Berners-Lee as an order-0 concept and Profession as an order-2 concept, it becomes apparent that the former cannot be a direct instance of the latter. Likewise, a specialization relationship between an order-1 concept Scientist and an order-2 concept Profession is equally unsound with respect to a set-theoretic interpretation of the model fragment.

Having to manually assign order values to each model element would be onerous; however, even in the absence of such information, the scenario in Figure 1 can still be identified as making unsound claims based on its inconsistent relationships. The "instance-of" relationship between Scientist and Profession is necessarily incompatible with the simultaneous claim that the former is a subtype of the latter, regardless of the absolute order values associated to these concepts. There is an inherent contradiction in instantiation requiring the two order values to differ by one and specialization requiring them to be the same.

The above explains the utility of "Ontological Anti-Patterns" that can be used to detect such ill-formed scenarios [6, 18]. In contrast to such a pattern-based scheme, the approach underlying MdM was not arrived at by mining data for problematic patterns; rather its well-formedness constraints originate from the motivation to ensure that models have a sound set-theoretic interpretation.

2.2. Previous Attempts

An early attempt to exclude ill-formed user models in the context of modeling with multiple levels of abstraction is "strict metamodeling" [8, 21]. Based on a single simple principle, it rules out a huge class of conceptual errors, including those characterized by the anti-patterns used by [18]. The downside of its very

conservative nature is that it also forbids users from adequately modeling a number of naturally occurring domain scenarios. The latter force users of "strict metamodeling" to employ workarounds that lead to "unnatural" solutions [22, section 8.1] or a duplication of elements [23], which not only add complexity of their own but also necessitate the introduction of additional constraints.

Many approaches aim to avoid the aforementioned downsides by using various concepts. The one most founded on ontological correctness is Almeida et al.'s MLT* which supports the adequate modeling of more demanding domain scenarios through the use of *orderless types* [24, 25]. While a disciplined use of the approach retains sanity-checking abilities for a large proportion of a user model, the remaining part, involving orderless types, cannot be fully checked anymore. Some users may hence unintentionally exploit orderless types to create unsound models, thus undermining the rigor that MLT* otherwise supports.

It could be argued that MDM similarly provides users with an escape route of simply adding yet another modeling dimension, whether justified or not. However, first, a user can never make a claim that a type classifies two elements with different orders while belonging to the same sub-domain as these elements. A user trying to use another modeling dimension simply as a workaround and then attempting to still establish a common root, will be prevented from doing so by various MDM constraints. Second, more often than not, introducing a different dimension will result in an accurate model. To the best of our knowledge all examples that have been given for orderless types, actually are best understood as involving two different modeling dimensions.

Note that MDM targets the modeling of natural user domains, such as those formed by physical objects occurring in the real world. To model linguistic domains, e.g., the OMG's MOF with its dual nature as a top-layer and representation format [26], mechanisms like orderless types could be considered to be more suitable, but a full discussion of the merits of orderless types vs orthogonal ontological classification is out of scope for this article.

2.3. Orthogonal Ontological Classification

"Multi-Dimensional Modeling" (MDM) based on the notion of "Orthogonal Ontological Classification" claims to fully retain the sanity-checking capabilities of "strict metamodeling" without incurring its downsides (cf. Section 2.2), while avoiding exploitable loopholes [12]. It claims to retain the same rigor for local hierarchies, referred to as "classification clusters", and argues that inter-cluster relationships cannot give rise to conceptually ill-formed models. It addresses challenging scenarios in which elements are ostensibly classified by multiple classifiers of different order (cf. Figure 2(b)), by maintaining that such overlapping classifications are best understood as occurring from different separate dimensions (cf. Figure 2(c)), using a "separation of concerns" approach.

Figure 2(a) shows a normal multi-level modeling hierarchy in which Susan's type, i.e., Corgi is typed by a higher-order type Dog Breed. Figure 2(a) shows potencies as opposed to absolute order values (cf. [27]), however, for our purposes here, we may assume that the potency values correspond to order values, i.e., that Susan is a pure object.

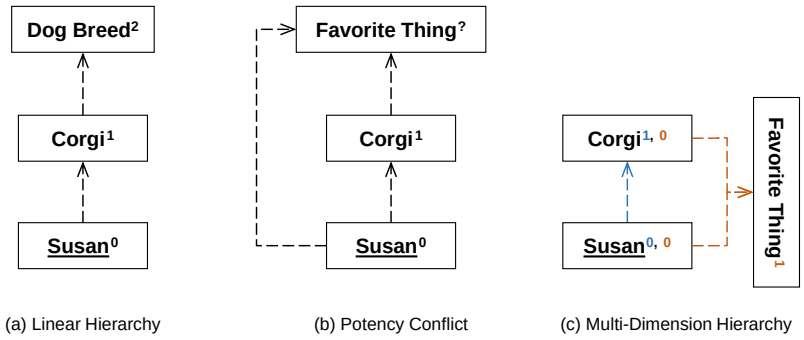


Figure 2: Traditional vs Orthogonal Classification

Figure 2(b) features a concept Favorite Thing that would be considered an *orderless type* in MLT* [24, 25], since it has both an order-zero instance Susan and an order-one instance Corgi. Hence, the potency of Favorite Thing is shown as "?" since its *classic potency* (cf. [28]) would have to be either 1 or 2 for Corgi and Susan respectively but cannot accommodate both cases at the same time. Figure 2(b) resolves this issue by considering Favorite Thing as classifying the two other concepts from a different dimension. With respect to this dimension (note the use of a different color for the respective instance-of relationships), Favorite Thing can now consistently be a potency-one classifier for two potency-zero concepts. Note Susan's two zero potency values which have the same color as the respective instance-of relationships.

The reason for the scenario shown in Figure 2(b) being unable to support a consistent order/potency to Favorite Thing is that it forces all three elements into a single linear classification hierarchy. Therefore, under "strict metamodeling" this scenario would be deemed as illegal whereas MLT* would require one to declare Favorite Thing as an orderless type. MDM resolves the tension on Favorite Thing by acknowledging that the latter does not extent the Susan-Corgi hierarchy (a proper extension would be DogBreed, cf. Figure 2(a)), but rather represents an orthogonal classification on both Susan and Corgi. It is worth pointing out that even though Susan has multiple classifiers, this scenario is not an example of traditional multiple classification in which both classifiers belong to the same classification hierarchy, e.g., as in an amphibious vehicle being classifiable simultaneously as a car and a boat, with both of the latter belonging to a domain of vehicles. In Section 5 we present an example model making use of MDM, demonstrating the latter's utility for organizing model content.

Although the original description of MDM [12] was focused on precise well-formedness criteria for models and elaborated on a number of constraints to be enforced, it was informal and had no implementation to back up its ideas. We therefore set out to investigate the suitability of CONCEPTBASE for realizing an MDM implementation, both in terms of the expressiveness of its specification language and the efficiency of its optimized deductive database engine.

3. Formalization

Our formalization of MDM in CONCEPTBASE is realized in O-TELOS (see Section 4), but for better accessibility we present a technology-independent formalization in this section. It covers a deliberately restrictive version of MDM (as outlined in [12, section 4.3]) which includes *characterization potency* [28]. We do not cover element features (e.g., attributes), since our focus is on validating MDM's main principles, which concern element relationships. In the following presentation of our formalization we typically refrain from elaborating the underlying rationales for the well-formedness conditions; these can be found in [12]. Whenever we deviated from the informally described well-formedness conditions of this source, we state our motivation and reasoning.

The model structures we are concerned with are graphs over elements. These "elements" are sometimes referred to as "clabjects" [8], because they can play the role of a class or an object, or both at the same time. These elements are connected with relationships, of which we only cover classification and generalization relationships here, as other relationships kinds are not restricted by MDM. Since elements have potencies (cf. [29]) that belong to dimensions (cf. Figure 2 & [12]) and classification/generalization relationships belong to dimensions as well, we use the many-sorted signature of equation 1.

$$\Sigma = (E, D, A, \rho) \tag{1}$$

$$E = \{e_i \mid e_i \in \mathit{Elements}\} \tag{2}$$

$$D = \{d_i \mid d_i \in \mathit{Dimensions}\} \tag{3}$$

$$A = \{a_i \mid a_i \in \mathit{Attributes}\} \tag{4}$$

$$\rho = \{:_d, \prec_d, .\} \tag{5}$$

We denote an element e with potency p in dimension d as e^{p_d}. If e_2 is a direct classifier of e_1 in dimension d, we use $e_1 :_d e_2$. If e_2 is a direct generalization of e_1 in dimension d, we use $e_1 \prec_d e_2$. Note that both ":" and "$\prec$" refer to respective abstract syntax relationships occurring in a model, i.e., their semantics is defined by a model interpretation which is defined separately, i.e., independently of syntactic well-formedness constraints. We use the combination $e_1 \dot{\prec}_d e_2$ as a shortcut for representing that e_2 is either a direct classifier or a direct generalization of e_1. This shortcut is used in the formalization only, i.e., MDM does not support a respective modeling relationship. We use a "$^+$" superscript notation to denote sequences of relationships of at least length one. For instance, $e_f :^+ e_l$ represents the scenarios $e_f : e_l$, $e_f : e_1 : e_l$, $e_f : e_1 : e_2 : e_l$, etc. Building on "$:_d$" and "$\prec_d$", $e_1 \,\overline{:_d}\, e_2$ applies whenever $e_1 :_d e_2 \;\vee\; \exists\, e_3 \;\; e_1 :_d e_3 \;\wedge\; e_3 \prec_d^+ e_2$, i.e., "$e_1 \,\overline{:_d}\, e_2$" applies if e_1 is directly or indirectly classified by e_2 in dimension d.

In the following, we use labels for our well-formedness constraints that correspond to the labels C_1-C_4 suggested in [12, section 4.3]. Since the latter do not cover characterization potency, we use a C_0 prefix for our respective constraints.

The first characterization potency constraint C_{0a} covers two aspects: First, upon instantiation potency must decrease (not necessarily exactly by one [28]), and second, only non-zero potency elements can be instantiated.

Instances must have a potency that is strictly lower than that of their classifiers and classifiers must have potencies greater than zero.

$$\forall\, e_{1,2}, p_2, d \quad e_1 :_d e_2^{p_{2\,d}} \;\rightarrow\; \exists\, p_{1\,d}\;\; e_1^{p_{1\,d}} \;\wedge\; 0 \leq\; p_{1\,d} < p_{2\,d} \qquad (\mathrm{C}_{0a})$$

Note the omission of any potency information on e_1 in the premiss, since we want to support underspecification of element potency. If we had formulated constraint C_{0a} to use $e_1^{p_{1\,d}}$ in the premiss already then the constraint would have only applied to instance-of relationships where both elements feature explicit potency values. Our approach here ensures that all instance-classifier pairs, where the classifier specifies a potency, obey the rules of characterization potency.

The second characterization potency constraint C_{0b} requires that subtypes must not have a potency that is lower than the supertype potency.

Subtypes must have an equal or higher potency than their supertypes.

$$\forall\, e_{1,2}, p_{1,2}, d \;\; e_1^{p_{1\,d}} \prec e_2^{p_{2\,d}} \;\rightarrow\; p_{1\,d} \geq p_{2\,d} \qquad (\mathrm{C}_{0b})$$

Constraint C_1, informally described and named *disjoint feature sets* in [12], is designed to avoid having to disambiguate access to element features in case multiple classifiers of an element define a feature with the same name. This is not an MDM-specific requirement but needs to be in place for any language that does not provide disambiguation mechanisms for resolving duplicate feature (e.g., attribute) contributions from multiple sources (classifiers and generalizations). In the constraint definition below, the pattern $(e_1.a) : e_2$ represents the existence of an e_1 feature a with type e_2.

Elements that classify or generalize the same element, must not define features with the same name.

$$\begin{aligned} &\forall\, e_{0,1,2,3,4}, a_{1,2} \\ &\quad (e_0 \,\overline{:}\, e_1 \;\vee\; e_0 \prec^+ e_1) \;\wedge\; (e_0 \,\overline{:}\, e_2 \;\vee\; e_0 \prec^+ e_2) \wedge \\ &\quad (e_1.a_1) : e_3 \wedge\; (e_2.a_2) : e_4 \;\wedge\; a_1 = a_2 \;\rightarrow\; e_1 = e_2 \end{aligned} \qquad (\mathrm{C}_1)$$

Note that even though we allow to optionally exclude multiple generalization within one dimension (via constraint C_{3b}), we still need to account for the potential presence of multiple generalization in the above constraint, since a user may not make use of constraint C_{3b}, or multiple generalization may occur involving supertypes from different dimensions. Although the latter is technically prohibited for this particular MDM language design by constraint C_{2c}, it can be a good idea to redundantly prevent certain scenarios to make the constraint system as a whole more robust against changes to individual constraints that may result in undesirable consequences.

Constraint C_2, named *bottom-level overlapping* in [12], ensures that there is a unique dimension, which an element that is classified from multiple dimensions can be instantiated into, thus obviating the need to specify a dimension when instantiating such an element. Note that here, "*bottom-level*", does not refer to an absolute bottom level of a hierarchy, but merely to the bottommost possible (potency-zero) position of elements in instantiation chains. We cover this aspect with our constraint C_{2a}.

Elements with potencies in more than one dimension must not have more than one non-zero potency value.

$$\forall\, e_0, p_{1,2}, d_{1,2}\ \ e_0^{p_{1\,d_1}} \wedge\ e_0^{p_{2\,d_2}} \wedge\ d_1 \neq d_2\ \wedge\ p_{1\,d_1} > 0\ \ \rightarrow\ \ p_{2\,d_2} = 0 \qquad (\mathrm{C}_{2a})$$

Note that potency values of zero prevent instantiation (cf. constraint C_{0a}), and that we do not specify a classifier for e_0 since we want to allow for e_0 to be a top-level element with a manually assigned potency. We are thus deviating from the "*bottom-level* overlapping" focus of the original C_2 constraint since we deemed that the constraint was in essence about preventing the potential of instantiation into more than one dimension, as opposed to only achieving this for elements that have explicit classifiers.

Furthermore, since it is possible to omit potency specifications for the purposes of under-specification, an element could potentially be instantiated into multiple dimensions without constraint C_{2a} preventing such a scenario, since the latter only covers cases featuring explicitly specified potencies. Constraint C_{2b} below addresses this by ensuring that elements are actually only instantiated into one dimension only, even in the absence of any potency information.

All instantiations from a classifier must be into the same dimension.

$$\forall\, e_{1,2,3}, d_{1,2}\ \ e_1 :_{d_1} e_3\ \wedge\ e_2 :_{d_2} e_3\ \rightarrow\ d_1 = d_2 \qquad (\mathrm{C}_{2b})$$

The next constraint, C_{2c}, is not part of the MDM well-formedness suggestions, however, we decided that a complement to constraint C_{2a} was useful that addresses the reception of type facets through specialization. Since constraint C_{2a} restricts elements to have a type facet from one dimension only, it makes sense to prohibit elements from receiving type facets via specialization from any other dimension.

Elements participating in multiple dimensions must not entertain generalization relationships in their potency-zero dimensions.

$$\begin{aligned}
&\forall\, e_{1,2}, d_{1,2} \\
&\quad e_1 \prec_{d_1} e_2\ \wedge\ \mathrm{member}_{d_2}(e_1) \wedge \\
&\quad d_1 \neq d_2\ \rightarrow\ \exists p_{d_1}\ \ e_1^{p_{d_1}} \wedge\ p_{d_1} > 0 \\
&\textit{where}\ \ \mathrm{member}_d(e) = \\
&\qquad \exists\, e_1\ \ (e :_d e_1\ \vee\ e_1 :_d e\ \vee\ e \prec_d e_1\ \vee\ e_1 \prec_d e\ )
\end{aligned} \qquad (\mathrm{C}_{2c})$$

Note that the "member"-predicate does require elements to have an explicit potency in a dimension. Dimension membership is solely acquired via respective relationships. This supports the underspecification of potency values, while simultaneously allowing checking for inappropriate type facet acquisition from dimensions that an element cannot be instantiated into anyhow.

Constraint C_{2c} works in tandem with constraint C_{2a}; the latter restricts the non-zero potencies of an element to a single dimension, so by demanding that there is a non-zero potency value for a dimension that an element entertains a generalization relationship with, no generalization in another dimension may exist in which the element has a non-zero potency.

The C_3 constraint, named *connected classification clusters* in [12], requires all elements within a dimension to form a monolithic "classification cluster", i.e., it prohibits classification clusters, constituted by instanceOf relationships, that belong to the same dimension, but are not connected to each other via classification relationships. In our formalization, constraint C_{3c} covers this aspect, however, since in a previous version of our formalization ([15]) we additionally had constraints C_{3a} & C_{3b}, we briefly discuss them here.

Constraint C_{3a} prohibited multiple classification within one dimension.

Elements must not have more than one classifier within a dimension.

$$\forall\, e_{0,1,2}, d \;\; e_0 :_d e_1 \;\wedge\; e_0 :_d e_2 \;\;\rightarrow\;\; e_1 = e_2 \qquad (C_{3a})$$

We introduced this constraint since, at the time, we did not have a ConceptBase formalization of the "*level-respecting*" property (cf. constraint C_{4c}). Because we since addressed this shortcoming, we no longer need to enforce constraint C_{3a}, as it is technically too strong, i.e., prohibits some sound models. Instead of dropping constraint C_{3a} altogether, we replaced the constraint with a query that users can optionally use to check for occurrences of multiple classification, in case they want to rule it out or just detect it for some reason.

The same reasoning applies to constraint C_{3b} which is about multiple generalization, rather than multiple classification. Although the original C_3 formulation of [12] does not imply that multiple generalization (aka, "multiple inheritance") needs to be prohibited, we defined a constraint C_{3b}, in the same vein as constraint C_{3a}, thus allowing users to optionally prohibit or just detect multiple generalization. Again, our current implementation does not enforce the constraint, but allows optional invocation in the form of a query.

Elements must not have more than one supertype within a dimension.

$$\forall\, e_{0,1,2}, d \;\; e_0 \prec_d e_1 \;\wedge\; e_0 \prec_d e_2 \;\;\rightarrow\;\; e_1 = e_2 \qquad (C_{3b})$$

The main "monolithic cluster" aspect of the original C_3 constraint is taken care of by our constraint C_{3c}:

Dimensions must not lack a classification root.

$$\begin{array}{l} \forall\, d \;\; (\exists\, e_1, e_2 \;\; e_1 :_d e_2) \;\rightarrow \\ \qquad (\exists\, e_5 \;\; (\forall\, e_{3,4} \;\; e_3 :_d e_4 \;\;\rightarrow\;\; e_3 :_d^+ e_5)) \end{array} \qquad (C_{3c})$$

An earlier version of this constraint did not feature the initial check for the presence of any elements participating in an instantiation relationship within a dimension, which led to a model containing a dimension declaration but otherwise being empty to be reported as violating constraint C_{3c}. Since an empty model trivially satisfies the requirement about the absence of disjoint classification clusters, we augmented the formalization of constraint C_{3c} accordingly.

If constraint C_{3a} and constraint C_{3c} are enforced in combination, they ensure that each dimension has a single unique classification root. Now that our implementation no longer enforces constraint C_{3a}, constraint C_{3c} on its own only ensures that there is at least one element from which all other elements

participating in classification relationships can be (deeply) instantiated from. In other words, there is no longer a requirement that this element constitutes a single unique root; other such roots may also exist. We decided that this is appropriate in contexts in which multiple classification is supported, since the main property – the absence of any isolated classification clusters within a dimension – is still ensured by constraint C_{3c}. In particular, a classification forest comprising disjoint trees is never allowed.

Note that constraint C_{3c}, which actually realizes the C_3 constraint of [12], is not about rejecting unsound models, but in effect rather about discouraging modelers from using orthogonal classification gratuitously, perhaps in ill-conceived attempts to avoid the order consistency enforcing rules that apply within a single classification dimension. Demanding that all classifiers within a dimension are joined via relationships, will cause modelers to reconsider their modeling choices, in case they cannot create a single classification cluster for each dimension unless they establish relationships between classifiers that are clearly unrelated to each other. As a result, constraint C_{3c} is not enforced by our implementation either, but is, like the other C_3 constraints, meant to support the optional checking of models for properties that some modelers may care about.

The final constraint C_4 of [12], named *sound meta-hierarchies*, concerns general well-formedness requirements that would apply outside a multi-dimensional approach as well and correspond to—in spirit but not as restrictively—the regiment established by "strict metamodeling" [8].

First, the graphs implied by models must be free of cycles with respect to classification and generalization relationships [16].

The graph of instanceOf and specializationOf relations must be acyclic.

$$\forall\, e\ \neg(e \dot{\prec}^{+} e) \qquad (C_{4a})$$

Through the use of "$\dot{\prec}^{+}$" we require every path with mixed classification and generalization relationships edges to be acyclic, as opposed to only imposing the constraint on pure classification and pure generalization paths respectively. Unlike the original C_4 constraint suggests, we do not restrict the context of the constraint to a single dimension only. In combination, these two choices lead to the rejection of a wider range of models with circular definitions, which we consider to be ill-formed, according to the spirit of the original C_4 constraint.

Second, an important component of the original C_4 constraint is that generalization relationships must not occur between elements of different order (i.e., of different set-theoretic classification power).

Elements in a generalization hierarchy must not have inconsistent orders.

$$\forall\, e_{1,2}, d\ \ \text{specConnected}_d(e_1, e_2)\ \rightarrow\ \text{orderNotInconsistent}_d(e_1, e_2) \qquad (C_{4b})$$

$$\begin{aligned} &\textit{where}\ \ \text{specConnected}_d(e_1, e_2) = \\ &\qquad (e_1 \prec_d e_2\ \vee\ e_2 \prec_d e_1)\ \vee \\ &\qquad (\exists\, e_3\ \ (e_1 \prec_d e_3\ \vee\ e_3 \prec_d e_1)\ \wedge\ \text{specConnected}_d(e_3, e_2)) \end{aligned}$$

$$
\begin{aligned}
&\text{orderNotInconsistent}_d(e_1, e_2) = \\
&\quad (e_1 = e_2) \vee \\
&\qquad \neg(\delta_d^+(e_1, e_2) \ \vee \ \delta_d^+(e_2, e_1)) \wedge \\
&\qquad (\exists\, e_3 \ \ \delta_d^+(e_1, e_3) \ \wedge \ \delta_d^+(e_2, e_3) \ \ \rightarrow \\
&\qquad (\forall\, e_{4,5} \ \ \delta_d(e_1, e_4) \ \wedge \ \delta_d(e_2, e_5) \ \ \rightarrow \\
&\qquad\qquad \text{orderNotInconsistent}_d(e_4, e_5)))
\end{aligned}
$$

$$
\begin{aligned}
&\delta_d(e_1, e_2) = \\
&\quad (e_1 :_d e_2) \vee \\
&\quad (\exists\, e_3 \ \ e_1 :_d e_3 \ \wedge \ \text{specConnected}_d(e_3, e_2)) \vee \\
&\quad (\exists\, e_3 \ \ \text{specConnected}_d(e_1, e_3) \ \wedge \ e_3 :_d e_2) \vee \\
&\quad (\exists\, e_{3,4} \ \ \text{specConnected}_d(e_1, e_3) \ \wedge \ e_3 :_d e_4 \wedge \\
&\qquad\quad \text{specConnected}_d(e_4, e_2))
\end{aligned}
$$

In the above, predicate specConnected_d holds if two elements participate in the same generalization/specialization hierarchy. Predicate $\text{orderNotInconsistent}_d$ holds in the absence of any relationships that imply order difference between the two elements in question.

We chose the name "orderNotInconsistent", rather than "orderConsistent", because we never aim to a) establish actual order values, nor do we ever b) prove that two order values are the same. Our approach is to only look for explicit information in the model that implies that the order values of two elements cannot be the same. Many models should be considered incomplete and it may just be the absence of an as of yet to be added relationship that prevents recognizing that one concept cannot possibly have the same order as another one. Outside a closed world assumption or having access to semantic definitions, it is therefore not possible to ever positively confirm that the modeler intended two elements to have the same order.

This approach of proving a negative is useful for MDM, which supports underspecification of element order, and for any other modeling language which may not feature an explicit "order" concept. We never require, assume, or compute any absolute order values, we only establish implied relative order relationships and check them for inconsistencies. Predicate $\text{orderNotInconsistent}_d$ achieves this by using the δ relation which we could have named "orderProgeny". The relation $\delta_d(e_1, e_2)$ holds, if $\text{order}(e_1) = \text{order}(e_2) - 1$. The relationship definition (cf. constraint C_{4b}) satisfies this equation since it considers all patterns in which e_1 is one instantiation step removed from e_2.

Third, the final component of establishing *sound meta-hierarchies* is to ensure that classification hierarchies are "*level-respecting*" [16]. The latter property guarantees the absence of classification relationships that imply conflicting order values for a single element (cf. Figures 1 & 2). As a consequence, elements can be given a consistent level assignment, without requiring any level-jumping, thus eliminating the lack of a sound set-theoretic interpretation of the classification hierarchy. While constraint C_{4b} aims at detecting order inconsistencies between elements in the same generalization hierarchy, constraint C_{4c} aims at detecting inconsistencies within a classification hierarchy.

Common classifiers of an element must not have different orders.

$$\forall\, e_{0,1,2}, d \;\; e_0 \,\overline{:_d}\, e_1 \;\wedge\; e_0 \,\overline{:_d}\, e_2 \;\rightarrow\; \text{orderNotInconsistent}_d(e_1, e_2) \qquad (\mathrm{C}_{4c})$$

Note that the only way in which alternative instantiation paths between two elements can be created is through the use of multiple classification. In a pure tree-shaped classification hierarchy, it is not possible for any one element to receive conflicting information about its order due to inconsistent classification depths. This is why our constraint considers elements which have more than one classifier. The constraint requires the respective classifiers to not engage in relationships that imply that their order must be different. It does so via predicate $\text{orderNotInconsistent}_d$ whose operands must (cf. the definition of constraint C_{4b}) either be the same element, or

1. are not (possibly deep) instances of each other, and
2. if they have a common (possibly deep) classifier then their paths to that common classifier have equal length (only considering "instanceOf" relationships as steps).

It is worth noting that constraint C_{4c}, by virtue of building on our δ relation, is not only considering pure classification hierarchies, but rather detects inconsistencies in hierarchies with a mixture of classification and generalization relationships. Hence, beyond checking consistency of pure instantiation path lengths, the predicate actually checks for general order congruence.

4. Implementation

The motivation for realizing the technology-independent formalization of MdM of the previous section using ConceptBase was fourfold:

1. the ability to apply constraints to numerous test models and example scenarios provided a means to validate that our formalization captured the intended meaning.
2. the implementation enabled empirical validation as to whether the claims associated with MdM can be substantiated.
3. we wanted to investigate to what extent the MdM principles can be supported by ConceptBase.
4. measuring the implementation's performance allowed insights into the feasibility of using ConceptBase to support MdM and other languages embodying similar principles.

4.1. ConceptBase *and* O-Telos

ConceptBase [30] is a deductive database system for managing models and metamodels. Its data model is based on the O-Telos language [31, 32] and its predicative specification language is based on Datalog with negation [33].

The specification language uses a closed world assumption and guarantees terminating evaluations of –

rules predicates that can infer information, similar to PROLOG rules,

constraints model integrity conditions which must always be satisfied, and

queries supporting the derivation of instances of so-called query classes.

Around 30 rules and constraints in CONCEPTBASE define the O-TELOS semantics for instantiation, specialization, attribution, and relationships. O-TELOS is similar to the OMG's MOF, in that O-TELOS can both be used to (in an extended variant or as is) directly represent user models, or to support the definition of modeling languages, which in turn are used to represent user models [26].

4.2. Module Architecture

The implementation of MDM comprises three CONCEPTBASE modules which contain object definitions (called "facts" in logic), rules, constraints and queries. Figure 3 shows the module hierarchy. Definitions in super-modules are visible in sub-modules, allowing a user model to benefit from all definitions starting with MultiDim up to System.

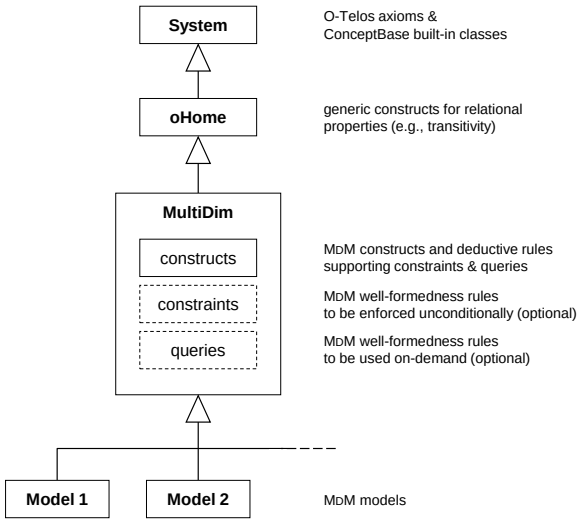


Figure 3: ConceptBase module architecture for the MDM implementation

All facts and rules pertaining to MDM, which are contained in module MultiDim, are therefore extending existing CONCEPTBASE facts and rules and provide the context in which user models can be defined and evaluated.

Users have a choice over using *constraints* within MultiDim, in case they want to guarantee that their models cannot violate any well-formedness rules, or, alternatively, use *queries*, in case they want to interactively and selectively check well-formedness aspects of their models.

ConceptBase compiles the respective construct-, constraints-, and queries-source files into its internal Datalog format, i.e. into Datalog facts and rules. All source files for the modules are available from our MDM project website [34].

4.3. Realizing Multi-Dimensional Modeling with O-TELOS

We had to make a fundamental decision whether to build on O-TELOS's definitions for instantiation and specialization, or to define a new language definition with custom instantiation and specialization relationships. We opted for the first alternative for the following reasons:

- MDM's classification and generalization notions are compatible with O-TELOS as far as the scope of our MDM specification is concerned,
- it minimizes effort, allowing us to focus on MDM-specific rules, and
- it supports a seamless adoption of MDM principles to O-TELOS.

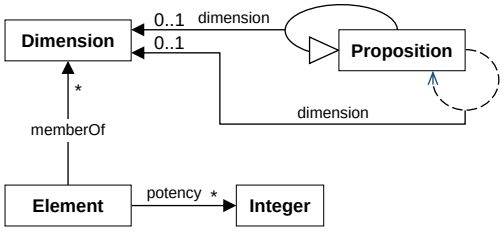


Figure 4: Structural embedding of MDM in O-TELOS

Figure 4 shows how we structurally embedded MDM, with its "clabject"-concept, named Element, into the O-TELOS data model. Elements can be members of multiple dimensions and have multiple potencies, one for each dimension they participate in. The classification and specialization relationships (defined at the most general class Proposition) also receive an optional dimension property.

With Element, we introduced an MDM-dedicated concept—which represents the notion of a Clabject, i.e., a concept that can be an instance, a type, or both at the same time [8, 29]—instead of building on the O-TELOS Individual concept, since we learned early on in our experiments that subjecting all O-TELOS objects to the MDM well-formedness principles resulted in undesirable constraint evaluation performance. This is due to MDM's inclusion of classification well-formedness and the fact that around 50% of predefined facts in ConceptBase are classification-related. We therefore confined the application of MDM-specific constraints to MDM-specific elements by letting respective quantified variables in the constraints range over Element rather than the O-TELOS Individual (cf. Sections 6 & 7.2.2 on the performance of our implementation).

We used a total of 26 ConceptBase rules to support the definition of the constraints listed in Section 3, including the definition of the relations in Equation 5, e.g., `directInstanceOf/lab` and `directSpecializationOf/lab`. Note the

use of an additional `lab` (short for "label") rather than a `dim` (short for "dimension") property. This reflects the fact that the dimension properties attached to these relationships are dimension *labels*, i.e., the names of dimensions. The latter are user-defined and our validation scenarios include labels such as `Products`, `Favorites`, `Activities`, `Assets`, etc. These are labels of the explicit dimension objects Products, Favorites, etc., they are linked to (cf. Listing 2).

The "member" predicate used in constraint C_{2c} (see Section 3) is defined by two mdrules, one of which is shown in Listing 1, with the other one analogously taking care of specialization relationships.

```
1 $ forall inst/InstanceOf x/Element dim/Dimension
2     (inst dimension dim) and (From(inst, x) or To(inst, x))
3     ==>  (x memberOf dim) $
```

Listing 1: Rule mdrule1

The O-Telos class `InstanceOf` referenced in line 1 of Listing 1 classifies all explicit instantiation relationships. Likewise, `Dimension` classifies all dimension objects. With `(inst dimension dim)` we establish that the `inst` relationship is linked to a `dim` dimension object.

The next two premisses (line 2 of Listing 1) establish that element "x" participates in the instantiation relationship `inst`. From these, it follows (in line 3) that element "x" is a member of dimension `dim`.

The rule in Listing 2 maps O-Telos instantiation to the MDM qualified instantiation predicate `(x instanceOf/lab c)`, where `lab` is the label of the dimension in which the instantiation is defined. An analog rule exists for specialization.

```
1 $ forall inst/InstanceOfWithDim x,c/Element lab/Label dim/Dimension
2     (inst dimension dim) and Label(dim,lab) and
3     From(inst,x) and To(inst,c)
4     ==>  (x directInstanceOf/lab c) $
```

Listing 2: Rule mdrule3

Of the nine constraints we defined, we show our implementation of constraint C_{2c} in Listing 3, since it

- shows a usage of the custom-defined `memberOf` predicate (cf. Listing 1).
- exhibits the slight implementation inelegance of having to deal with both dimension objects and dimension labels.
- is one of the richer constraints but still nicely demonstrates how readable ConceptBase constraints are.

```
1 c2c: $ forall x,c/Element lab/Label dim/Dimension
2          (x directSpecializationOf/lab c) and
3          (x memberOf dim) and not Label(dim,lab)
4          ==>  exists p/Integer (x potency/lab p) and (p > 0) $
```

Listing 3: Constraint C_{2c}

Figure 5 shows a model which is rejected due to violating constraint C_{2c}. Here, Corgi is not allowed to inherit a type facet from Person while also receiving a type facet from DogBreed; having two type facets from different dimensions is not supported by the minimal MDM language design we adopted from [12].

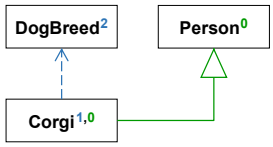


Figure 5: C_{2c}-violating model

If a user triggers constraint C_{2c} by extending the model, the constraint's custom error message (see Listing 4) is output by ConceptBase, along with an explanation detailing which elements are involved in violating the constraint.

```
1 MultiDimRules!c2c with
2   comment
3     hint: "Elements participating in multiple dimensions
4     must not entertain generalization relationships
5     in their potency-zero dimensions!"
6 end
```

Listing 4: Custom C_{2c} error message

Listing 5 shows how constraint C_{4b} (see Section 3), can be concisely formulated in ConceptBase by making use of rules.

```
1 c4b: $ forall x,y/Element lab/Label
2          (x specConnected/lab y)  ==>  (x orderNotInconsistent/lab y) $;
```

Listing 5: Constraint C_{4b}

Line 2 in Listing 5 specifies that if an element "x" is in the same generalization hierarchy as another element "y" then those two elements must not have inconsistent order values. The symmetric and transitive `specConnected` relationship is concisely defined by rules mdrule17 & mdrule18, each rule fitting in one line. The predicate `orderNotInconsistent`, which is also used by constraint C_{4c}, is defined by two rules, mdrule25 & mdrule26, requiring two and four lines of definition, respectively.

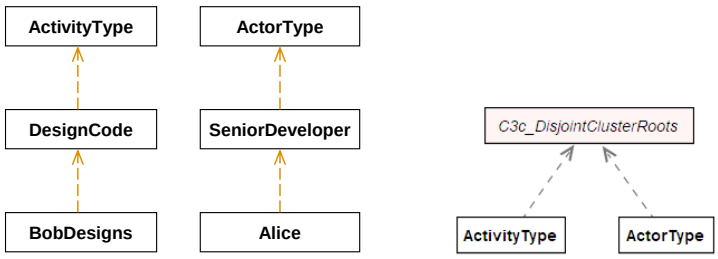
We separated constraint definitions from rule definitions by using different files since it can be desirable to not enforce constraints, e.g., to allow intermediate, not well-formed editing states during development, or when defining negative validation examples. We were thus able to only include constraints if and when we wanted, e.g., to confirm that models passed or failed validation.

4.4. C_3 constraints as queries

ConceptBase allows the specification of constraints, which are always enforced, and/or queries, which can be run on demand (see Section 6). We have implemented all constraints of Section 3 in both constraint and query form, to give users both options, depending on preference.

Our default set of ConceptBase constraints excludes the three C_3 constraints of Section 3 and provides them as queries only. Our reasons for doing so are: a) these constraints do not rule out unsound models (cf. Section 3), and b) their nature suggests modelers will be interested in being visually pointed to violation occurrences (see Figure 6), rather than receiving violation reports.

Further research is needed concerning the merit of constraint C_{3c} since non-adherence clearly does not make a model unsound. However, running the respective query may highlight disjointed classification clusters that might be indicative of an indiscriminate use of orthogonal classification. Modelers would thus be prompted to double check their modeling choices. If all highlighted applications are correct, there will be no challenge in connecting them via generalization or classification. In some cases, if the latter is not possible, the initial assumption that they all belong to the same classification dimension can be corrected by separating subclusters into multiple clusters, each belonging to its own distinct classification dimension. Figure 6(b) shows a ConceptBase screenshot in which our query representing constraint C_{3c} was used to identify multiple classification cluster roots for a Process dimension in the model shown in Figure 6(a). In this particular case, the hierarchies should be modeled using different dimensions (e.g. Activities vs Actors) rather than combining them to constitute a single cluster.



(a) Disjoint classification cluster (b) Query Result

Figure 6: Running a query in ConceptBase

4.5. ConceptBase *Visualization Support*

Beyond supporting the representation of MdM models and allowing them to be checked against well-formedness rules, we also implemented some visualization support. Element attributes and potencies are rendered below an Element's name, within the same visual element, instead of the standard ConceptBase approach that visualizes every attribute on its own and draws links between them and their owning elements.

Note the colored relationships in the ConceptBase screenshots (Figures 6 & 7). Modelers may specify RGB colors to be used to highlight dimension membership for relationships or backgrounds for dimensions such as Animals (see Figure 7 and Listing 6).

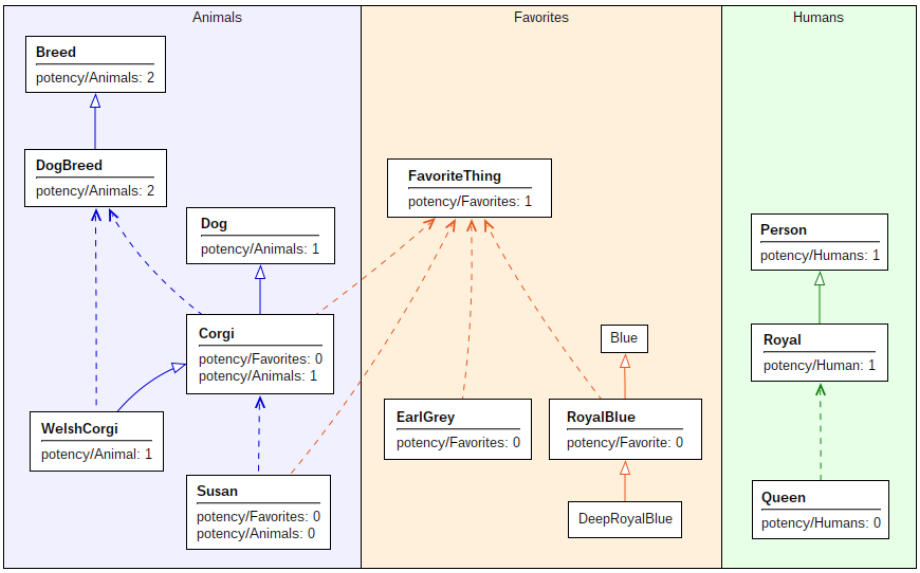


Figure 7: Screenshot: Explicit dimensions in ConceptBase

```
1 Animals in Dimension with
2   dimcolor col: "0,0,210"
3   gproperty bgcolor: "240,240,255"
4 end
```

Listing 6: Example dimension definition

Listing 7 shows one of the rules in the MultiDim module (cf. Figure 3), that implement the coloring of instanceOf links.

```
1 colorrule1: $ forall inst/InstanceOf dim/Dimension col/String
2   (inst dimension dim) and (dim dimcolor col)
3   ==> (inst gproperty/edgecolor col) $;
```

Listing 7: Rule for assigning colors to instanceOf links

The visualization support of dimensions in the form of colored backgrounds has, apart from providing a visual representation of the actual dimension objects in the model, only a presentational role. Figure 7 shows an example model featuring three dimensions. We omitted associations and links for clarity but note that these could have crossed dimension boundaries (cf. Figure 10). Two elements, Corgi & Susan, have classifiers in two dimensions and could have been placed on the boundary between the dimensions they participate in. We chose to place both elements solely into the Animals dimension, since that is where their essential (as opposed to secondary) classifiers are, without making any claims that this is a general guideline that should be followed.

All implementation source files and some of the models we used for validation purposes—in source format but also as PNG files—are available at https://purl.org/cbmdm [34].

5. An Example Model: Digital Twins

This section introduces an example model to a) illustrate the organizational qualities of MDM, b) further demonstrate the CONCEPTBASE implementation, and, c) communicate the basis of our performance measurements (see Section 6).

With respect to the first purpose, we decided to pick a domain that naturally features multiple modeling aspects and is suitable to illustrate modern development practices. Production technology has evolved from *mass production* (a single product model with many instances), to *mass customization* (a product model with variation points with many instances per variant), and more recently to *mass individualization* (each instance is described by a singleton type which specializes upper level product types) [35]. For instance, the majority of all new trucks models by Mercedes-Benz are custom-designs, i.e., are significant modifications of a base model [36]. This trend to mass individualization coincides with an increased usage of so-called digital twins. The latter are representations of physical artifacts and both can appear in the same model, e.g., to support the use of 3D modeling and simulation tools. Many artifacts in such domains depend on each other, e.g. a software component on a simulation model. We therefore chose the satellites domain as our example domain, taking inspiration from [37]. Our choices were furthermore informed by the involvement of one of the authors in a virtual engineering project, where the provenance of digital artifacts is recognized [38]. Beyond dependency and provenance information, artifacts may also feature ownership information. We therefore not only model satellite representations, of both physical satellites and their digital twins, but also artifact kinds and owner kinds, using MDM to separate these concerns.

5.1. Classification hierarchies and attributes

The partial model in Figure 8 shows the classification and generalization relationships of our example model. The left hand side features the satellite hierarchy (within dimension Satellites) including representations of physical and virtual satellite instances (GioveA1_r1 & GioveA1_v1 respectively). The top concept SatelliteModel has potency two (i.e., a maximum instantiation depth of two) and defines an attribute plannedmass. The subclass GalieoModel adds two more attributes. Its instance, GioveA1, is a particular Galileo satellite model but is also characterized as a digital artifact by virtue of being an instance of DigitalArtifact from the Artifacts dimension. Since GioveA1 is a digital artifact itself, rather then being a classifier for digital artifacts, it has potency zero with respect to that dimension and provides values for the version and lastupdate attributes. In contrast, in the satellites dimension, GioveA1 has potency one (as it is a classifier for satellite instances) and provides (type level-) values for the plannedmass, plannedorbit and frequency attributes. The two instances of GioveA1, i.e., GioveA_r1 and GioveA_v1, represent a physical satellite and its digital twin respectively. Only the latter is also characterized as a DigitalArtifact and specifies values for the respective version and lastupdate attributes. Note how the elements within the Satellites dimension that are also classified by elements in the Artifacts dimension, i.e., GioveA1, GioveA_r1 and GioveA_v1, are at different classification levels

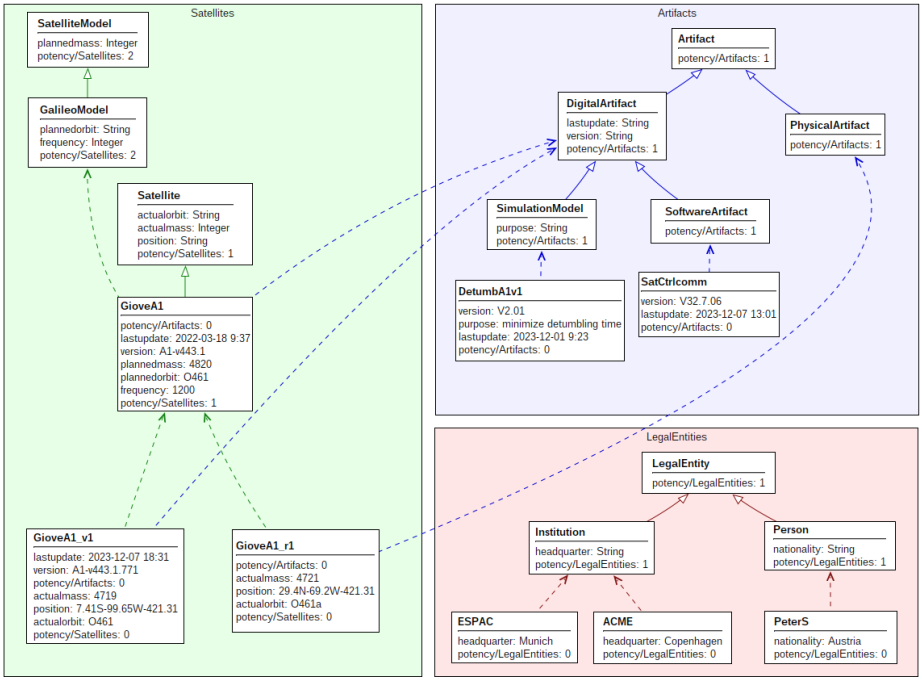


Figure 8: Satellite model hierarchies and dimensions in ConceptBase

within the Satellites dimension, but share the same classification level (zero) with respect to the Artifacts dimension. They also have in common that they provide values for attributes from both dimensions.

The Artifacts dimension comprises more elements beyond DigitalArtifact to support a fine-grained categorization of domain artifacts. For instance, SoftwareArtifact has an instance SatCtrlcomm, to be used as part of the control software of a satellite, and SimulationModel has an instance DetumbA1v1 which is employed to minimize the detumbling time of satellites.

Figure 9 shows the result of running constraint C_{3c} in its query form. As can be gathered either from the textual list on the left or the graphical representation on the right, there are two classes in dimension LegalEntities and four classes in dimension Artifacts that have been identified as local roots within a dimension that does not comprise a single classification cluster, i.e., a classification hierarchy in which all classifiers are connected to a unique root. Challenging a modeler to come up with such a unique root per dimension may help the modeler to confirm that all classifiers in a dimension form a coherent cluster, as opposed to being a loose collection of disjointed concerns. In this example model, this could be accomplished by adding ArtifactType and LegalEntityType respectively to the affected dimensions and declaring the reported elements as their instances respectively.

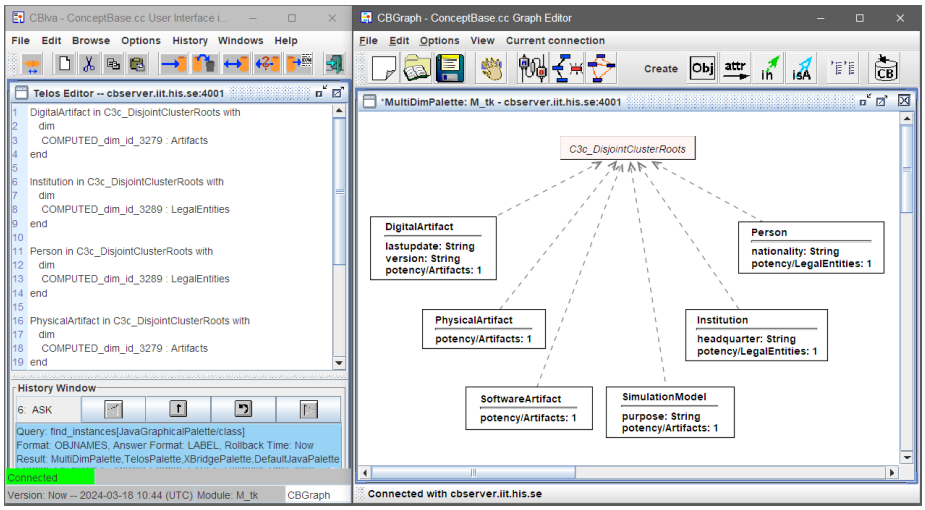


Figure 9: Running the C3c_DisjointClusterRoots query on the satellite model

5.2. Relationships

Although we could have included relationships in Figure 8 already, we chose to present them separately for clarity (see Figure 10). The embedded association between Satellite and Software Artifact enables satellite representations and software artifacts to be linked by embedded links. See GioveA_r1 & GioveA_v1 being linked to SatCtrlcomm, reflecting the fact that the software is embedded in both the physical satellite and its digital twin.

Likewise, the owner association between Artifact and LegalEntity enables artifacts to be linked to their owners. For instance, both SatCtrlcomm and DetumbA1v1 are owned by PeterS. Note that ownership can not only be expressed between individual satellite representations and their owners, but also between a specification, such as GioveA1, and its owner, the institution ACME.

Finally, the reflexive association depOn of Artifact allows one to specify a dependency network between artifacts [39], e.g., a dependency between SatCtrlcomm and DetumbA1v1. Likewise, the simulation model DetumbA1v1 depends on the digital twin GioveA_v1, as it uses its mass value as a simulation parameter.

Overall, our example model illustrates how the MDM approach supports the separation of various concerns—satellite modeling, artifact organization, and ownership—ensuring that each concern is free of a large class of structural soundness issues, while allowing each concern to be applied to any of the others, including future uses to other domains. We furthermore demonstrate how MDM supports a single relationship kind, e.g., concerning ownership, to be defined only once, but be soundly applied across multiple levels within a dimension, such as Satellites. Links between elements, may, but do not have to, cross dimension borders, depending on where the link participants are located. See, for instance, the intra-dimension link depOn between SatCtrlcomm and DetumbA1v1, vs the inter-dimension link depOn between DetumbA1v1 and GioveA1_v1.

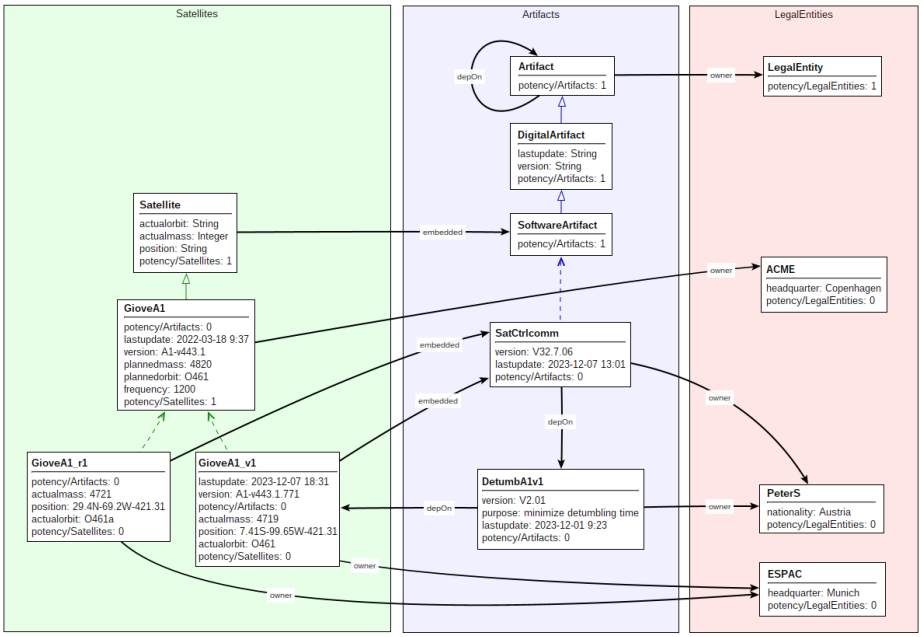


Figure 10: Satellite model relationships in ConceptBase

6. Performance

Constraint execution efficiency was far from a priority for us but we nevertheless investigated

- whether ConceptBase's deductive database engine is sufficiently performant to support the validation of approaches like MdM, i.e., those that imply the checking of global model properties.
- to what extent constraint evaluation times differ from each other, in order to see whether that would reveal anything about the nature of the constraints or untapped optimization potential of ConceptBase.
- how ConceptBase's various evaluation alternatives compare to each other in terms of evaluation speed, when MdM constraints are subjected to MdM models.

Note that the following results are specific to ConceptBase, i.e., they should not be construed to have any bearing on results one may expect from an equivalent formalization in other systems, such as Flora-2 [40] or XSB-Prolog [41].

Although we have made various performance measurements throughout development, in this section we focus on the results of evaluating the example model from Section 5 since it was the most suitable to highlight performance differences between constraints and/or the evaluation alternatives supported by ConceptBase.

ConceptBase offers three choices for evaluating well-formedness conditions:

E_{c1}: constraints (e.g., Listing 8) are loaded before a model is loaded/created. Each part (i.e., fact) of a model is then incrementally evaluated against the constraints. A constraint is only evaluated, if the addition or removal of a model fact may influence its evaluation. This is the standard approach when using ConceptBase and makes evaluation of constraints mandatory.

E_{c2}: constraints are added after the model has been loaded/created. Each constraint is then evaluated for the whole model. This alternative is not well-supported by ConceptBase since once constraints are loaded, they are mandatorily applied upon subsequent changes to the model, i.e., the approach effectively reverts to E_{c1}, after the constraints have been added.

E_q: checks are run as queries (e.g., Listing 9) on demand. Unlike for E_{c1} & E_{c2}, if queries are used to detect well-formedness issues, models may violate well-formedness conditions at any point in time. Queries are only ever run if and when requested by the user.

E_{c1} is best for users for whom well-formedness of models is of paramount importance at all times and who like to be informed of constraint violations at the earliest time possible. E_{c2} also always evaluates all constraints, but only at a point of the user's choosing and may in some cases be a quicker alternative for checking complete models (cf. Table 1). E_q provides users with the most flexibility, allowing them to specifically target certain properties when required, e.g., after a modeling phase has been completed.

Listings 8 & 9 show how we implemented constraint C_{2a} as a ConceptBase constraint and query respectively.

```
c2a: $ forall x/Element p1,p2/Integer lab1,lab2/Label
         (x potency/lab1 p1) and (x potency/lab2 p2) and
         (lab1 <> lab2) and (p1 > 0)  ==>  (p2 = 0) $
```

Listing 8: C_{2a} as a ConceptBase constraint

While Listing 8 shows a straightforward translation of constraint C_{2a}'s formal definition, the query in Listing 9 deviates from a direct translation in two aspects, so that it can report all non-compliant elements:

```
C2a_MultipleNonZeroPotency in QueryClass isA Element with
  constraint
    multinonzero: $ exists p1,p2/Integer lab1,lab2/Label
      (this potency/lab1 p1) and (this potency/lab2 p2) and
      (lab1 <> lab2) and (p1 > 0) and (p2 > 0) $
end
```

Listing 9: C_{2a} as a ConceptBase query

First, in contrast to the constraint, the query expression has to identify a violation, as opposed to describing the well-formed case; note the different quantification required, the change from an implication to a conjunction, and compare `(p2 = 0)` of the constraint to `(p2 > 0)` within the query.

Second, the query expression has to identify which of the elements involved is to be identified as becoming part of the query result set. In this case, it is element "x" of Listing 8, i.e., an element with more than one non-zero potency, that needs to be reported. This identification is accomplished by the use of "**this**", which is implicitly existentially qualified and in this case replaces every occurrence of "x" in the constraint variant.

Both evaluation alternatives E_{c2} and E_q may potentially benefit from cached evaluation results that were created when evaluating the same or a different constraints/queries earlier. In contrast, E_{c1} may only benefit from cache results that are produced while evaluating constraints in response to the addition or removal of a single model fact. To allow for a better comparison of the evaluation alternatives, we therefore evaluated constraints individually, as opposed to appliying them as a set, and cleared the cache for E_{c2} & E_q, before each individual run. As a result, the total figures at the bottom of Table 1 are the sums of individual evaluation times, not the total time needed when evaluating all constraints in combination. The evaluation times[1] of Table 1 where obtained using the satellite model of Section 5.

Table 1: Evaluation times for the satellite model (in seconds)

Constraint	E_{c1}	E_{c2}	E_q
C_{0a}	0.02	0.30	0.01
C_{0b}	0.01	0.21	0.01
C_1	1.48	0.63	0.99
C_{2a}	0.01	<0.01	<0.01
C_{2b}	0.30	0.63	0.57
C_{2c}	0.18	0.20	0.19
C_{3a}	0.02	0.32	0.29
C_{3b}	0.01	0.21	0.18
C_{3c}	0.05	0.02	0.34
C_{4a}	0.51	0.54	0.50
C_{4b}	1.69	0.55	0.53
C_{4c}	1.47	0.77	0.69
Total	5.75	4.38	4.30

While E_{c1} is generally fast and often beats the alternatives, it takes moderately longer for C_{4b} and C_{4c}, both of which imply order consistency checking. We make further observations regarding implementation performance in Section 7.2.2.

[1] Measurements made using an Intel Core i9-11950H.

7. Discussion

In the following, we first discuss our formalization choices and the consequences resulting from them. In Section 7.2 we then discuss the merit of ConceptBase as a supporting tool.

7.1. Formalization Discussion

The MDM article our work is based on [12], proposed four informally described C_1–C_4 "constraints" that outline a minimalistic realization of the approach. They are rather constraint categories, each of which typically requiring multiple formal constraints to be covered, hence the use of our sublabels $_a$, $_b$, etc. Whenever constraint definitions called for precision that was not detailed in the informal descriptions or generalizations seemed logical, we were often able to improve or expand on the informal design. In particular, we

- generalized the exclusion of classifier feature clashes (C_1) to include supertype feature clashes (cf. constraint C_1).
- generalized the prohibition to instantiate a "bottom-level" element participating in multiple dimensions into more than one dimension (C_2), to include elements at the top level. The latter may entertain potencies from multiple dimensions and should not be instantiatable into multiple dimensions either (cf. constraint C_{2a}).
- not only exclude cycles within classification- and generalization hierarchies respectively (C_4), but also exclude cycles comprising mixed relationships of the former kinds (cf. constraint C_{4a}).
- explicitly support dimension underspecification, i.e., allow dimensionless relationships and/or elements without explicit potencies.

We retained the absence of explicit "levels" and element-"order" notions from the original design, which makes the approach agnostic to the presence or absence of such notions in the model, and even the language specification. In particular, we infer order differences from classification relationships without relying on pure classification hierarchies, i.e., interspersed generalization relationships do not interfere with the determination of relative order values. This design elegantly targets the root cause of soundness violations all the same and could be regarded as avoiding overspecification in comparison to a level-based approach. However, this also obviously means that we do not support explicit "order" assignments or "level" allocations which could be checked for consistency with the rest of the model. On the plus side, this makes the approach straightforwardly adoptable to any technology that does not rely on explicit level or order values.

7.1.1. Generality

Compared to our initial attempt at covering C_4 from [15], constraint C_{4b} now detects a considerably larger class of ill-formed models, by covering indirect "instance Of" relationships as well. Furthermore, via an improved version of constraint C_{4c}, we can now detect inconsistencies which are caused by instantiation paths to a shared element which differ in length. Again, these paths may include indirect classification, i.e., checking is not limited to incarnation trees/graphs only. These significantly improved versions cover a larger class of sources for order inconsistencies while still allowing a concise constraint formulation and efficient evaluation. Unlike in our earlier work, we therefore do not have to categorically rule out multiple classification within a dimension in order to make sure that classification hierarchies are "*level-respecting*".

In its original formulation [16], the "*level-respecting*" well-formedness condition implied that comparing the lengths of instantiation paths is required.

level respecting $\forall n, m :$
$$(\exists\, e_1, e_2 : e_1 R^n e_2 \wedge e_1 R^m e_2) \;\rightarrow\; n = m$$

In the above, "R" is a placeholder for a metarelation and in our case it translates to classification. The respective ConceptBase implementation would therefore have to look similar to the hypothetical code in Listing 10, assuming that the "^" notation allows one to capture the length of an instantiation path between two elements in a classification branch.

```
1 $ forall n, m/Integer exists e1,e2/Element
2   (e1 instanceOf^n e2) and (e1 instanceOf^m e2)  ==>  n = m $
```

Listing 10: Level-respecting

However, ConceptBase does not support specifying path lengths in the above manner. While some of our other constraints make use of transitive instantiation chains, the constraints are agnostic about the lengths of such chains. We addressed this challenge by observing that the requirement of identical path lengths can be transposed into a requirement about the existence of a synchronized traversal of both paths. There is no need to actually determine the number of edges in paths, since it is sufficient to require that both paths can be traversed in lock-step, starting from one element (`e1`) and arriving, with the same final step, at the shared classification ancestor (`e2`). This strategy allowed us to formulate predicate orderNotInconsistent$_d$ as a ConceptBase rule and thus use the latter in constraint C_{4c}. We therefore successfully settled the hitherto unanswered question whether ConceptBase affords sufficient expressiveness to formalize the "*level-respecting*" requirement.

The advantage for modelers is that our implementation now allows them to use multiple classification within a dimension, but only if it does not introduce soundness issues by involving "diamond-shaped" classification hierarchies with inconsistent path lengths, as the latter would imply conflicting order values for an element at the bottom of such a diamond structure.

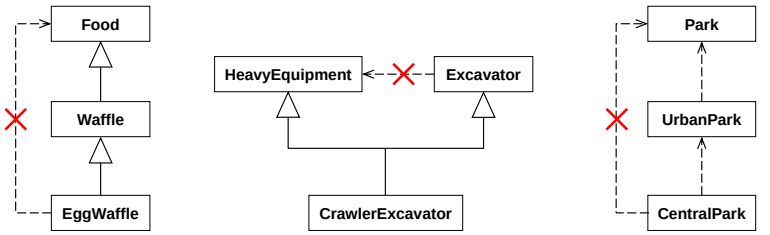


Figure 11: Wikidata Anti-Pattern Scenarios (cf. [18, Fig. 6, Fig. 8, Fig. 10])

7.1.2. MDM *Well-Formedness vs Anti-Patterns*

In [18], Brasilero et al. analyze Wikidata content with respect to problematic model fragments. Without fully exploiting their underlying axiomatic theory, they formulate three anti-patterns as SPARQL queries to quantify certain modeling issues in Wikidata. Given that these anti-patterns were derived independently from the MDM well-formedness principles, we were interested in whether our formalization and implementation would subsume the detection abilities of the anti-patterns AP1–AP3 presented in [18].

Figure 11 shows three Wikidata model fragments, which exemplify violations captured by the three anti-patterns Brasilero et al. used. In general, anti-patterns represent schemata, i.e., will detect a wide range of unsound model fragments, not just specific scenarios. For instance, the generalization hierarchy involving Food and EggWaffle in Figure 11(a), could involve arbitrarily many generalization relationships; as long as a classification relationship between these two elements exists, the model is not sound.

Our well-formedness rules cover all the scenarios detected by anti-patterns AP1–AP3, plus many more. On close inspection, it is clear why this has to be the case: There is a single reason as to why AP1 & AP2 appropriately reject offending models: elements that are connected via generalization relationships, must have the same order; otherwise, no sound set-theoretic interpretation of the respective model exists. Our constraint C_{4b} simultaneously covers AP1 & AP2, and more anti-patterns that could be formulated, since it targets the root cause that underlies the validity of these anti-patterns, i.e., the fact that a classification relationship between elements in the same generalization hierarchy is irreconcilable with the need for these elements to have the same order. Unlike AP1, for instance, constraint C_{4b} also rejects models like that in Figure 11(a) where the classification relationship is reversed, i.e., where Food is declared to be an instance of EggWaffle, or where a classification relationship exists between any elements in the generalization hierarchy, not just the bottom and the top ones. Likewise, constraint C_{4c} takes care of AP3 violations in a very general manner: it ensures that all instantiation paths between any two elements have the same path lengths. This means that not only single-step shortcuts (like between CentralPark and Park) are excluded, but any inconsistent combination, such as three steps vs four steps to a shared ancestor, etc.

We observed that the anti-pattern scenario variations we considered in our constraint validation process sometimes violated more than one constraint. This appears to testify to a useful robustness quality of a sanity-checking approach that covers multiple soundness principles. For instance, the aforementioned variation of AP1 in Figure 11(a) (with the classification relationship inverted), violates both constraints C_{4a} & C_{4b}.

It should be noted, that the anti-patterns used in [18, 19] were solely used as queries to search for certain ill-formed model fragments and hence should not be judged as attempting to comprehensively cover potential integrity violations or be regarded as reflecting the capabilities of MLT*. These anti-patterns were only inspired by the latter's respective axiomatic theory. However, regardless of the original intent for using anti-patterns, we note that for the purposes of ensuring the well-formedness of models, an approach based on constraints, which embody fundamental soundness principles, seems more suitable than a collection of schemata that were devised on the basis of found integrity violations. The model in Figure 12 supports this conjecture.

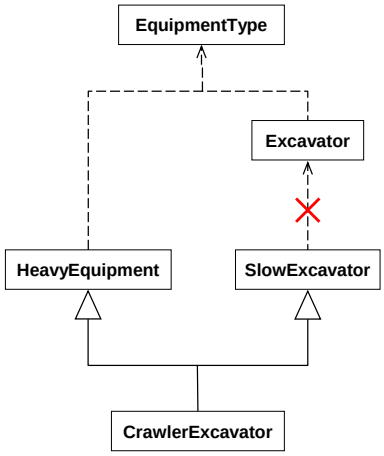


Figure 12: Unsound model invisible to anti-patterns AP1–AP3

The "instanceOf" relationship marked with the red cross in Figure 12 would imply that SlowExcavator has a different order than HeavyEquipment (which shares the same order with Excavator) and thus make it unsound for CrawlerExcavator to specialize both HeavyEquipment and SlowExcavator. Our C_{4b} constraint detects this inconsistency but none of the anti-patterns AP1–AP3 do.

Overall, we did not attempt to capture a full-fledged language design that addresses all possible design choices, e.g., we did not impose limitations on specializations into multiple dimensions. Our only intention was to capture the constraint categories C_1–C_4 of [12].

7.2. Implementation Discussion

The motivation for creating an implementation (see Section 4) included the aim to understand how well ConceptBase can support a language design like MdM. In the following we discuss the merit of ConceptBase for this task and also briefly comment on how the availability of an implementation supported our formalization efforts (see Section 7.2.5).

7.2.1. Expressiveness

It is remarkable how close ConceptBase constraints (e.g., Listing 5) are to a concise logic formulation (cf. C_{4b}). There is some contamination of the pure logic with some realization details due to the need to distinguish between dimension objects and their corresponding labels but overall the ConceptBase constraints are very readable and effortlessly supported experimentation.

Adequately supporting the "*level-respecting*" property without limiting user models (cf. Section 7.1.1) required a second iteration but once we found the right approach, the implementation was very straightforward.

7.2.2. Efficiency

ConceptBase's evaluation of rules and constraints is not sufficiently efficient to support all MdM constraints at the O-Telos level, at least not when using E_{c1}(see Section 6). We therefore let our constraints range over a dedicated Element class instead of the O-Telos class Individual and let some rules range over DimensionLabel instead of Label—the former matches only MdM dimension lables, wheres the latter matches all labels, including O-Telos labels—to achieve practical evaluation times. Employing Element and DimensionLabel, rather than Individual and Label, reduced the number of possible values for certain variables, and hence the size of the search space, without changing the semantics of the formulas. The only downside of this optimization is that not all MdM well-formedness constraints cannot readily be applied to O-Telos models in general, since the latter are not using Element and DimensionLabel.

Given these domain restrictions, our small validation scenarios are checked very swiftly. Checking medium-sized models, like the example model of Section 5, does not complete instantly but does not take longer than 4 to 6 seconds for all twelve constraints depending on the evaluation strategy.

These performance results were partly achieved by adding a new optimization rule to the ConceptBase code. Prior to this optimization, the generated code for some constraints used a suboptimal sequence of join operations of instantiation/specialization predicates.

Without the aforementioned optimizations, some constraints took several minutes to complete with the E_{c1} method. Overall, a speedup factor between 10 and 20, depending on the evaluation method, specifically for constraints C_1, C_{4b}, and C_{4c} was achieved. For example, the evaluation of constraint C_1 against the complete MdM model of Section 5 initially required 160 seconds and is now completed in 1.48 seconds. The results confirm that even approaches like MdM, that imply checking global model properties, can be efficiently implemented with a deductive system such as ConceptBase.

7.2.3. Usability

We paid a limited amount of attention to usability concerns since our emphasis was on exploring the feasibility of formalizing MDM constraints using CONCEPTBASE, as opposed to creating a feature-complete, working environment. Optimizing model loading times, allowing model elements to be highlighted as problematic on demand (rather than outright rejecting the model as ill-formed), and advanced visualization features, such as rendering potencies as superscript values, therefore were not priorities for us. Overall, we treated CONCEPTBASE mainly as a user model storage backend, as opposed to an environment with extensive support for interactive modeling.

Having said that, we support visual presentation of dimensions (cf. Figures 7, 8, and 10), color-coding of relationships with respect to the dimensions they participate in, and a compact visualization of concepts with their properties. Our implementation inherits CONCEPTBASE properties, such as the ability to handle models of considerable size, create models textually or with a graphical editor, ensure the well-formedness of accepted models, and report any well-formedness violations with user-friendly messages.

7.2.4. Constraints vs Queries

As mentioned in Section 6, one may formulate a well-formedness condition as a CONCEPTBASE constraint or a CONCEPTBASE query. Since the regular modus operandi, when using CONCEPTBASE, is to define constraints in a module that is loaded before one loads or creates a new model, and constraints are unconditionally evaluated by CONCEPTBASE whenever a model element is added or changed, one straightforward approach to divide up well-formedness conditions into constraints and queries is to formulate those that are essential to model soundness as constraints. In our case only those constraints with a C_0 or C_4 prefix are absolutely required to ensure model soundness. In contrast, the constraints with a C_2 prefix are meant to enforce a simple MDM language design which avoids models whose semantics are of yet undefined. Constraints C_{3a} & C_{3b} allow users to remove multiple classification and/or generalization from the allowed modeling constructs, but since in this improved version of the formalization we were able to introduce constraint C_{4c} and have constraint C_1, those two C_3 constraints are no longer necessary to ensure model soundness. Constraint C_{3c} should be useful to motivate modelers to double check whether their dimensions are coherently defined (cf. Sections 3 & 4.4) but, like its C_3 siblings, does not constitute a necessary condition for model soundness.

Apart from performance considerations (cf. Section 6) and the above point on necessity of enforcement, there are two further criteria one may consider when deciding between expressing a well-formedness condition as a constraint or a query: Using queries, one is able to

1. work with unsound models, which can be useful during editing, creating and storing models that are supposed to violate well-formedness rules, etc.
2. identify model elements as the source of an issue by having them highlighted in the model editor.

When ConceptBase reports a constraint violation, it only states the names of involved model elements, whereas the result of a query can be made to visually highlight respective elements in the model.

Note, however, that this feature may also be regarded as creating an additional responsibility for the designer of a query. While Constraints C_{3a} & C_{3b} naturally translate into queries that report a minimal amount of elements that cause a violation, this is not the case for constraint C_{3c}. Starting from a formulation as a ConceptBase constraint (see Listing 11), the straightforward way of expressing it as a ConceptBase query would result in Listing 12. This, however, would result in all elements within a dimension to be reported, as long as the condition of a single classification cluster is not met.

```
c3c: $ forall lab/Label
         (exists u,v/Element (u instanceOf/lab v))  ==>
         (exists c/Element
            (forall x,y/Element (x instanceOf/lab y)
                                ==>  (x instanceOf_trans/lab c))) $;
```

Listing 11: C_{3c} constraint

```
C3c_DisjointElements in QueryClass isA Element with
   constraint
   disjointDim: $ exists x0/Element lab/Label
      ((x0 instanceOf/lab this) or (this instanceOf/lab x0)) and
      not (exists c/Element (forall x,y/Element
         (x instanceOf/lab y)  ==>  (x instanceOf_trans/lab c))) $
end
```

Listing 12: Straightforward C_{3c} query

Since being presented with every element participating in a classification relationship within a dimension that violates constraint C_{3c} would be rather overwhelming for the modeler and not constitute any information beyond that the classification structure in the dimension does not form a single classification cluster, we designed an alternative query (see Listing 13) that only returns the roots of the cluster fragments presented in the dimension. These are the elements that would have to be connected to a common classifier in order to address the violation, or, alternatively, would have to form unique roots in freshly created dimensions, in case the dimension under evaluation is deemed to be incoherent.

```
C3c_DisjointClusterRoots in GenericQueryClass isA Element with
  parameter, computed_attribute
    dim: Dimension
  constraint
    disjointRoots: $ exists lab/Label
      (this in TopInDimension[~dim/dim]) and Label(dim,lab) and
      exists x0/Element (x0 instanceOf/lab this) and
      (not exists c/Element (forall x,y/Element
        (x instanceOf/lab y)  ==>  (x instanceOf_trans/lab c))) $
end
```

Listing 13: C_{3c} query reporting offending roots only

Query `C3c_DisjointClusterRoots` thus not only allows modelers to judge cluster roots for coherence, but also produces a minimal set of elements that would have to be affected when establishing a single classification cluster.

While this variant is far from being a straightforward translation of the original constraint and requires the use of another `TopInDimension` query to solely consider classification roots as potentially offending elements, we believe it can be of significant value to modelers to receive such specific results as opposed to all the elements within a non-conforming dimension, or, alternatively, the name of a non-conforming dimension only. The particular version of `C3c_DisjointClusterRoots` in Listing 13 also allows generalization relationships to join cluster fragments, which deviates from constraint C_{3c} but which we present here as a refined interpretation of when a dimension cluster should be considered to be disjoint.

In order to afford users with utmost flexibility, we provide all well-formedness constraints as both ConceptBase constraints and ConceptBase queries [34]. Users can hence decide for themselves which subset of well-formedness conditions should be unconditionally enforced and which should be interactively and selectively checkable on-demand.

7.2.5. Utility for Formalization and Validation

ConceptBase proved to be very useful for validating our formalization. In many cases, subjecting select modeling scenarios to our constraints simply confirmed the latter's adequacy and/or the claimed properties of MDM. In some cases, however, ConceptBase supported experimenting with variants, e.g., to explore alternative formulations or achieve better evaluation efficiency. By defining a validation suite of model scenarios that target all constraints respectively, we were able to trial tweaks and either confirm or disprove that they were still reporting ill-formed models and not reporting sound models.

Our suggested amendment to the original C_2 formulation in the form of a slightly wider constraint C_{2a} definition was arrived at during such constraint validation experiments. While working with respective validation scenarios, it seemed suboptimal to forbid the clashing of multiple potency values greater than zero only for elements that have explicit classifiers (cf. Section 3).

The aforementioned validation suite also served as a way to empirically confirm that MDM's claims regarding the mechanical detection of unsound models are justified. We purposefully designed scenarios we deemed sound and others we deemed unsound for various reasons, using a rough 50:50 distribution, and our final formalization of MDM reliably produced the expected positive or negative results, including for scenarios we did not create ourselves, such as the anti-pattern examples. Internally, we have used well over 50 models, of which 39 are included in our published regression validation suite at [34]. Model sizes are typically very small, i.e., usually comprising not much more than six elements, but increasing element volume would not have added any further insights beyond confirming ConceptBase's ability to handle large models. We created the vast majority of models ourselves and only occasionally used third-party models, such as the anti-pattern examples.

8. Conclusion

The more critical the reliance on the conceptual integrity of a model is, the higher the need to eliminate avoidable conceptual mistakes. It is concerning that modeling concepts of societal importance, such as "gene", "protein", and "disease" are used inconsistently in models [19, Table 1]. Ontological sanity-checking of models is not a novel concept, but so far multi-level modeling users were either forced to complicate their models by having to work around overly strict well-formedness requirements, or were given mechanisms that circumvent otherwise strict rules, with the potential of allowing inadvertent misapplications with ill consequences.

In this article, we presented a formalization of an approach [12] that reliably and independently of the modeling domain prevents a large class of ill-conceived conceptualizations without requiring modelers to explicitly provide semantic descriptions of the concepts they are using, and without forcing them to work around unnecessary limitations imposed by overly strict well-formedness criteria when modeling naturally occurring domain scenarios.

Our formalization does not rely on explicit "order" or "level" constructs, making it widely applicable, i.e., a candidate for adoption by other multi-level modeling approaches. Compared to our earlier work, we have given users the option to use multiple classification within a modeling dimension without reducing the rigor of the soundness checking. We achieved this by ensuring the "*level-respecting*" property in the most general way. Additionally, we significantly improved the constraints on specialization hierarchies. Being based on fundamental soundness principles, rather than attempting to match ill-formed model fragments, our implementation is not dependent on a comprehensive schematic capturing of such fragments.

We have empirically validated MDM claims and our implementation by using numerous models of which the most essential form a regression validation test suite that we made use of any time we explored a constraint variant, e.g., to increase the scope of a constraint or improve upon its evaluation efficiency. In many cases we additionally created a systematic exploration of scenarios, in which, for instance, all combinations of relationship directions in specific scenarios were explored. Our slight modification of constraint C_{2a} and our introduction of constraint C_{2c} were the direct result of following such a tool-supported and scenario-based exploration approach.

By creating a sample model for a realistic, multi-level-, multi-aspect modeling domain, and running performance tests, we demonstrated that ConceptBase is capable of supporting a concise, intuitive and sufficiently performant implementation of MDM, while not requiring any coding at any stage. Although usability was not a priority, the implementation offers decent notation support, colored relationships, and explicit dimension containers.

We are convinced that our work is a suitable foundation for further exploration of the MDM paradigm, allowing richer variants to be examined and validated. Our formalization and public implementation open up these avenues for anyone who may want to extend or adopt the approach to fit their frameworks.

Acknowledgments

This work was in part supported by the Swedish Knowledge Foundation (KKS) through its VF-KDO Profile research project, grant number 20180011. We are grateful to the anonymous reviewers whose in-depth feedback led to considerable improvements.

References

[1] A. Pirotte, E. Zimányi, D. Massart, T. Yakusheva, Materialization: A powerful and ubiquitous abstraction pattern, in: Proceedings of the 20^{th} International Conference on Very Large Data Bases (VLDB'94), Morgan Kaufman, 1994, pp. 630–641.

[2] C. Partridge, S. de Cesare, A. Mitchell, J. Odell, Formalization of the classification pattern: survey of classification modeling in information systems engineering, Software & Systems Modeling 17 (1) (2018) 167–203. doi:10.1007/s10270-016-0521-5.

[3] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos: Representing knowledge about information systems, Information Systems 8 (4) (1990) 325–362.

[4] M. A. Jeusfeld, B. Neumayr, DeepTelos: Multi-level modeling with most general instances, in: Conceptual Modeling - 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, 2016, pp. 198–211. doi:10.1007/978-3-319-46397-1_15.

[5] C. Atkinson, T. Kühne, Rearchitecting the UML infrastructure, ACM Transactions on Modeling and Computer Simulation 12 (4) (2003) 290–321.

[6] G. Guizzardi, It's patterns all the way down: Ontological patterns, antipatterns and pattern languages for next-generation conceptual modeling, ACM Lecture (2020).
URL https://speakers.acm.org/lectures/13930

[7] C. Atkinson, T. Kühne, Reducing accidental complexity in domain models, Software and Systems Modeling 7 (3) (2008) 345–359. doi:10.1007/s10270-007-0061-0.

[8] C. Atkinson, Meta-modeling for distributed object environments, in: Enterprise Distributed Object Computing, IEEE, 1997, pp. 90–101.

[9] R. Gitzel, M. Merz, How a relaxation of the strictness definition can benefit MDD approaches with meta model hierarchies, in: Proceedings of the 8^{th} World Multi-Conference on Systemics, Cybernetics & Informatics, Vol. IV, 2004, pp. 62–67.

[10] J. de Lara, E. Guerra, R. Cobos, J. Moreno-Llorena, Extending deep metamodelling for practical model-driven engineering, The Computer Journal 57 (1) (2012) 36–58. doi:10.1093/comjnl/bxs144.

[11] U. Frank, Multilevel modeling - toward a new paradigm of conceptual modeling and information systems design, Business & Information Systems Engineering 6 (6) (2014) 319–337. doi:10.1007/s12599-014-0350-4.

[12] T. Kühne, Multi-dimensional multi-level modeling, Software and Systems Modeling 21 (2) (2022) 543–559. doi:10.1007/s10270-021-00951-5.

[13] B. Neumayr, M. A. Jeusfeld, M. Schrefl, C. Schütz, Dual deep instantiation and its ConceptBase implementation, in: Proceedings Advanced Information Systems Engineering CAiSE 2014, Springer Int. Publ., Cham, 2014, pp. 503–517.

[14] M. A. Jeusfeld, J. P. A. Almeida, V. A. Carvalho, C. M. Fonseca, B. Neumayr, Deductive reconstruction of MLT* for multi-level modeling, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '20, 2020, pp. 1–10. doi:10.1145/3417990.3421410.

[15] T. Kühne, M. A. Jeusfeld, Sanity-checking multiple levels of classification: A formal approach with a conceptbase implementation, in: Conceptual Modeling: 42nd International Conference, ER 2023, Lisbon, Portugal, November 6–9, 2023, Proceedings, Springer-Verlag, Berlin, 2023, p. 162–180. doi:10.1007/978-3-031-47262-6_9.

[16] T. Kühne, Matters of (meta-) modeling, Software and System Modeling 5 (4) (2006) 369–385. doi:10.1007/s10270-006-0017-9.

[17] C. Partridge, A. Mitchell, M. da Silva, O. X. Soto, M. West, M. Khan, S. de Cesare, Implicit requirements for ontological multi-level types in the uniclass classification, in: Companion Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '20, 2020, pp. 1–8. doi:10.1145/3417990.3421414.

[18] F. Brasileiro, J. P. A. Almeida, V. A. Carvalho, G. Guizzardi, Applying a multi-level modeling theory to assess taxonomic hierarchies in Wikidata, in: Proceedings of the 25th International Conference Companion on World Wide Web, WWW '16 Companion, Int. WWW Conferences Steering Committee, 2016, pp. 975–980. doi:10.1145/2872518.2891117.

[19] A. A. Dadalto, J. P. A. Almeida, C. M. Fonseca, G. Guizzardi, Type or individual? Evidence of large-scale conceptual disarray in Wikidata, in: Proceedings of Conceptual Modeling - 40th International Conference, ER 2021, Vol. 13011 of LNCS, Springer, 2021, pp. 367–377. doi:10.1007/978-3-030-89022-3_29.

[20] OMG, Unified Modeling Language Superstructure Specification, Version 2.1.1, OMG document formal/07-02-05 (Feb. 2007).

[21] C. Atkinson, R. Gerbig, Melanie: Multi-level modeling and ontology engineering environment, in: Proceedings of Modeling Wizards'12, ACM, 2012, pp. 1–2.

[22] A. Lange, C. Atkinson, Multi-level modeling with LML – A contribution to the multi-level process challenge, International Journal of Conceptual Modeling 17, special Issue: Multi-Level Process Challenge (Jun. 2022). doi:https://doi.org/10.18417/emisa.17.6.

[23] T. Kühne, A. 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, ACM, NY, USA, 2022, p. 434–443. doi:10.1145/3550356.3561571.

[24] J. P. A. Almeida, C. M. Fonseca, V. A. Carvalho, Comprehensive formal theory for multi-level conceptual modeling, in: Proceedings of 36th International Conference on Conceptual Modeling, Vol. LNCS 10650, Springer, 2017, pp. 280–294.

[25] C. M. Fonseca, J. P. A. Almeida, G. Guizzardi, V. A. Carvalho, Multi-level conceptual modeling: From a formal theory to a well-founded language, in: Proceedings of the 37th International Conference on Conceptual Modeling (ER 2018), LNCS 11157, Springer Verlag, 2018, pp. 409–423.

[26] C. Atkinson, T. Kühne, Concepts for comparing modeling tool architectures, in: L. Briand (Ed.), Proceedings of the ACM/IEEE 8^{th} MODELS, Springer Verlag, 2005, pp. 398–413.

[27] T. Kühne, A story of levels, in: Proceedings of the MODELS 2018 Workshops co-located with the 21^{th} ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'18), Vol. Vol-2245 of CEUR Proceedings, ISSN 1613-0073, 2018, pp. 673–682.

[28] T. Kühne, Exploring potency, in: ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18), ACM, 2018, pp. 2–12. doi:10.1145/3239372.3239411.

[29] C. Atkinson, T. Kühne, The essence of multilevel metamodeling, in: Proceedings of the 4^{th} International Conference on the UML 2000, Toronto, Canada, LNCS 2185, Springer Verlag, 2001, pp. 19–33. doi:10.1007/3-540-45441-1_3.

[30] M. A. Jeusfeld, Metamodeling and method engineering with ConceptBase, in: Metamodeling for Method Engineering, MIT Press, 2009, pp. 89–168. URL https://conceptbase.sourceforge.net/2021_Metamodeling_for_Method_Engineering.pdf

[31] M. A. Jeusfeld, What is O-Telos?, accessed 2024-03-04: https://conceptbase.sourceforge.net/O-Telos.html (1999).

[32] M. Koubarakis, A. Borgida, P. Constantopoulos, M. Doerr, M. Jarke, M. A. Jeusfeld, J. Mylopoulos, D. Plexousakis, A retrospective on Telos as a metamodeling language for requirements engineering, Requir. Eng. 26 (1) (2021) 1–23. doi:10.1007/s00766-020-00329-x.

[33] S. Abiteboul, R. Hull, Data functions, Datalog and negation, SIGMOD Rec. 17 (3) (1988) 143–153. doi:10.1145/971701.50218.

[34] M. A. Jeusfeld, T. Kühne, ConceptBase implementation of MdM, Project Web Site (2025).
URL https://purl.org/cbmdm

[35] S. J. Hu, Evolving paradigms of manufacturing: From mass production to mass customization and personalization, Procedia CIRP 7 (2013) 3–8, 46th CIRP Conference on Manufacturing Systems 2013. doi:10.1016/j.procir.2013.05.002.

[36] Mercedes-Benz, Mercedes-Benz information for bodybuilders, Webpage, last accessed: 28 March 2024 (2024).
URL https://trucks.mercedesbenzmena.com/leading-star/en/applications/bodybuilder/

[37] P. M. Fischer, D. Lüdtke, C. Lange, F.-C. Roshani, F. Dannemann, A. Gerndt, Implementing model-based system engineering for the whole lifecycle of a spacecraft, CEAS Space Journal 9 (2017) 351 – 365. doi:10.1007/s12567-017-0166-4.

[38] I. Morshedzadeh, A. H. C. Ng, M. A. Jeusfeld, J. Oscarsson, Managing virtual factory artifacts in the extended PLM context, J. Ind. Inf. Integr. 28 (2022) 100369. doi:10.1016/j.jii.2022.100369.

[39] M. Jarke, M. A. Jeusfeld, T. Rose, A software process data model for knowledge engineering in information systems, Inf. Syst. 15 (1) (1990) 85–116. doi:10.1016/0306-4379(90)90018-K.

[40] G. Yang, M. Kifer, C. Zhao, Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web, in: R. Meersman, Z. Tari, D. C. Schmidt (Eds.), ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003, Vol. 2888 of Lecture Notes in Computer Science, Springer, 2003, pp. 671–688. doi:10.1007/978-3-540-39964-3_43.

[41] XSB, XSB Prolog, Website, last accessed: 28 March 2024 (2023).
URL https://xsb.com/xsb-prolog/