Managing Model Complexity: A Segment-Based Approach

Thomas Kühne
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
tk@ecs.vuw.ac.nz

Pierre Maier
Faculty of Computer Science
University of Duisburg-Essen
Essen, Germany
pierre.maier@uni-due.de

Abstract

Multi-level modeling (MLM) is typically characterized as minimizing *accidental complexity*. We observe that MLM levels also aid in managing *essential model complexity* by separating and organizing model content based on the domain classification level it represents. Building on this insight, we generalize the notion of a level—and previously recognized pluralities of elements, e.g., those contained in packages—to that of a *model segment*. In addition to solution-based segments, we recognize domain-induced segments, such as *domain diversity*, allowing semantic information to be represented in models that was previously lost to mapping different modeler intents to the same modeling constructs. These new, semantically relevant, model segments support better model navigability and aid model understanding by facilitating the isolation of particular model aspects. Crucially, all model segments are subject to being represented by proxies, thus enabling a significant reduction of first-contact model complexity. We posit that introducing segments for managing model complexity has the potential to lower the threshold of MLM adoption due to its ability to counteract the often-held perception of MLM models being complex.

CCS Concepts

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

Keywords

multi-level, complexity, model management, model segments

ACM Reference Format:
Thomas Kühne and Pierre Maier. 2026. Managing Model Complexity: A Segment-Based Approach. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3837062.3838697

1 Introduction

Model complexity is commonly encountered in real-world models and represents a threat to model usability since complex models can be hard to understand, quality-control, and maintain. Contributing factors to model complexity are large element counts and ostensibly intractable relationship networks between those elements. Adverse effects of overly complex models include:

- users are facing *information overload*, making it difficult to identify relevant parts of the model.
- important structure is difficult to recognize, inviting model changes that inadvertently harm model integrity.
- navigation becomes unwieldy, imposing significant traversal efforts on users.

One kind of complexity is *accidental* in the sense that it can be eliminated by employing technological means [5]. For instance, the impedance mismatch between domains featuring multiple levels of classification and two-level technologies can be addressed by using multi-level modeling (MLM) [22]. Complexity that is not amenable to such remedies – i.e., is inherent to the subject under study – is referred to as *essential* [5]. While this kind of complexity cannot be eliminated, it can be managed by model decomposition, introducing abstractions, and using complexity-hiding model views.

We observe that MLM not only addresses accidental complexity but also supports managing essential model complexity by

- supporting the introduction of higher-level abstractions (like class diagrams extract and document the structure underlying complex object diagrams), and by
- distributing elements that would cohabit the same technical space when using two-level technologies, to multiple levels (in the same manner as layered architectures separate elements of a functionality stack into distinct layers).

Self [32] represents a language design which does not aim at either of the above. By adopting a prototype-based approach, i.e., entirely forgoing the explicit notion of classes, *Self* deemphasizes the difference between classes and objects, and while some structure (objects playing the role of classes) may be extractable by identifying particular types of cloning and delegation usages, that structure is not as explicit as it is in an MLM model.

Despite being undeniably more expressive, e.g., by affording more control [23], MLM models are often perceived as being more complex than two-level models, partly due to their use of richer notation. Any means for reducing the apparent complexity of MLM models and/or facilitating the management of their complexity would therefore be much welcome with respect to increasing the adoption of MLM in academia and industry.

In this paper, we introduce *model segments* in order to improve model complexity management and thus facilitate model understanding and maintainability. We first elaborate on what we mean by "model complexity" (Section 2), then introduce "model segments," including their causal origin and their representation (Section 3), and follow with proposals how to leverage model segments to improve complexity management (Section 4). We discuss the expected positive impact of adding segmentation to MLM (Section 5) before we conclude (Section 6).

This work is licensed under a Creative Commons Attribution 4.0 International License.
MODELS Companion 2026, Málaga, Spain
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2903-4/26/10
https://doi.org/10.1145/3837062.3838697

2 Model Complexity

There is no single commonly accepted definition of model complexity. Typically, model size and the number of inter-dependencies between elements are considered to be contributing factors to model complexity [17, 26]. It is not straightforward to design a complexity metric, since neither the nodes nor the edges in a graph representing a model are of the same kind and, as such, produce different amounts of cognitive load.

2.1 Characterization

In this paper, we informally assume model complexity to be analogous to software complexity, i.e., as being related to the level of difficulty to maintain a model [33, p. 1], noting that the latter requires understanding the model and the ability to change a model without unintentionally breaking it. In particular, we are not considering subjective model complexity, i.e., factors that depend on the model recipient, such as the level of familiarity with MLM constructs or MLM notation. Rather, we are concerned about modelers being overwhelmed with information overload, missing important structural constraints, and not seeing the "big picture" in general, e.g., due to unrealized opportunities for model decomposition [7].

In the following, we assume that model complexity can be tamed by supporting the modeler to focus on a reduced number of elements and/or relationships.

2.2 Common Management Approaches

There are numerous approaches intended to address essential complexity in one way or another. In this section we first look at commonly used approaches to then make the case for *model segments* as a foundational concept that is suitable to capture the management approach of MLM (see Section 1) and many other related complexity management techniques.

Commonly appearing complexity management approaches that often are supported at the language level include packages, subsystems, components, modules, layers, swimlanes, aspects, views, and design patterns. Oftentimes elements are

- hidden by some kind of container (e.g., packages, subsystems, components, modules),
- separated using a useful segregation principle (e.g., layers, aspects, swimlanes),
- filtered to provide a specific focus (e.g., views), or
- recognized to form a regular solution structure (e.g., design patterns [12]).

In each case, a reduction of many elements is achieved, thus helping to abstract away from a plurality of elements when either referring to what allows them to be grouped together is sufficient, or grouping them aids focusing on the elements in one group at a time, disregarding all elements in other groups.

Beyond the aforementioned, there are several further related approaches that propose solutions to reduce the number of elements in a model in order to improve the extensibility and reusability of models. Heinrich et al. [16] propose "metamodel modules" that can be organized in ordered "layers," where a lower layer serves to extend a higher layer. COLD (concern-oriented language development) [8], is based on a "concern" notion to manage language complexity. Unlike components, concerns account for variability by including a feature model. De Lara et al. [10] similarly suggest modeling language families through feature models. Language components are also supported by language workbenches such as MontiCore [29]. None of these approaches, however, account for multiple classification levels and they do not intend to provide a more generalized notion of groupings of model elements, nor do they attempt to distinguish various kinds thereof and relate them to the original modeling domain.

3 Model Segments

We posit it is useful to generalize the aforementioned complexity management approaches to a unified concept, identify what motivates incarnations of this concept, and separate the identification principle used to ascertain concept membership from how and when concepts are presented to a modeler (see Section 4).

We refer to this unified concept as a *model segment*. A segment

(1) includes elements that can be meaningfully grouped together according to some criterion.
(2) usually has sibling segments, i.e., it segments (divides up) many elements into multiple segments.
(3) typically allows its members to be confined to a convex spatial region in a diagram layout.

This general characterization not only captures the notions mentioned in Section 2.2, but also many more ways of carving up a large model into more manageable parts, e.g., MLM levels, generalization hierarchies (where subtypes can be represented by their respective supertypes), whole-part aggregations (where the aggregate can represent all its parts), classification dimensions [21], etc.

In the following, we identify segment kinds (Section 3.1), analyze what in the domain or solution structure may give rise to useful segmentation (Section 3.2), and finally discuss options for defining segments (Section 3.3).

3.1 Segment Kinds

An MLM hierarchy (see Figure 1; the dashed horizontal lines denote level boundaries) naturally achieves segmentation – the totality of elements in a model is divided up, often based on their classification order (cf. *order-synchronization* [20]) – and defines an orientation in the sense that model elements are divided up into vertically stacked areas, commonly referred to as *levels*. Segmentation may also occur horizontally, though. For instance, the three colored areas in Figure 1 (one of them containing two distinct shades of yellow) can be regarded as horizontally *dividing* the middle classification layer. There is no single partitioning principle governing all three areas at play here though, hence the use of different colors. The green segment, for instance, is based on aggregation, capturing all elements belonging to a particular whole-part structure.

Note that not every intra-level segmentation needs to result in horizontal division. The two shades of yellow in the middle level in Figure 1, for instance, show a vertical segmentation between a generalization and its specializations. The yellow segment (including both shades of yellow) is therefore subdivided vertically in the same orientation as the tall blue *slice* in Figure 1 is subdivided. However, unlike the latter, which is subdivided based on the degree of element order, it is subdivided based on the degree of generality. Note that both blue and yellow segments have a vertical as well

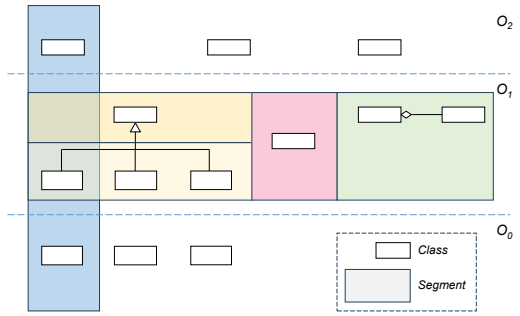

Figure 1: Outline of Prospective Segment Kinds in MLM

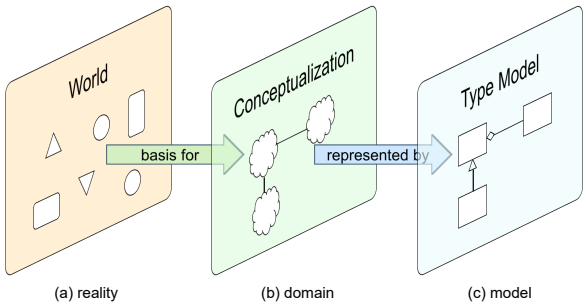

Figure 2: Domains as Conceptualized Reality

as a horizontal extent. Some segment kinds may be hierarchical in nature, i.e., support recursive nesting of the same segmentation principle. For instance, the green aggregation-motivated segment in Figure 1 could organize parts of parts into respective subsegments.

After having considered the general geometry of segments in MLM, we now turn our attention to potential sources of segmentation.

3.2 Segment Formation Sources

Motivation for model segmentation originates from two different source categories:

Domain-induced The modeling domain often implies ways to structure a model. Respective sources are language-independent, i.e., induce the same model segmentation regardless of the modeling language used.

Solution-induced Modeling languages often provide structuring mechanisms that aim at decomposing models for the purpose of obtaining manageable subdivisions, modularization, untangling responsibilities, etc.

Since modeling languages commonly provide aids for capturing domain-induced segmentation, characterizing a particular segmentation source is not always straightforward. We nevertheless think it is useful to distinguish between segmentation sources induced by the domain and language mechanisms that support model segmentation.

3.2.1 Domain-Induced Sources. Domains, or subjects under study, generally feature a rich internal structure that could be explicitly captured by respective segment kinds in models. We assume that modeling is a two-step process that starts with reality, proceeds to a conceptualization of said reality (which is colored by one's world-view) and then ends with a representation of said conceptualization in the form of a model which is expressed in some modeling language (see Figure 2).

For the purpose of this paper we assume that models target an object-oriented conceptualization of reality, which we refer to as the (modeling) *domain*, and that *domains* include abstractions, such as types, generalizations, aggregates and their parts, etc. While other definitions of *domain* are possible and other world views may be preferable in some applications, we maintain that it is justifiable to view classification, generalization, and aggregation as explicitly occurring in domains.

In general, we recognize the following domain-induced segment sources:

Time. Dynamic domains give rise to model elements that vary over time, inducing segments that capture time-stamped scenarios.

Abstraction Level. It is often useful to segregate model elements according to which domain abstraction level they correspond to, according to a particular world-view or conceptualization paradigm (cf. Figure 2). MLM exploits this for classification levels, but analogous segments could be used for generalization (cf. the yellow shades in Figure 1), or aggregation depth.

Note that depending on the level-organization principle used by an MLM language, domain-induced abstraction segments may or may not align with model level boundaries. For instance, with respect to all domain individuals, the corresponding segment would match the bottom level in an *order-synchronized* level hierarchy, but may not in a merely *order-aligned* hierarchy [20].

Figure 3 depicts three "classification level" segments in an MLM level hierarchy, using the same color coding for all three levels (rather than changing luminance depending on the level) to emphasize that they all belong to the same *level hierarchy* segment group. Figure 3 also shows an example of an aggregation segment at the bottom right. We based our examples on the *MULTI Collaborative Comparison Challenge* [27] and are using *deep connectors* [1] (cf. the black-diamond composition relationships) and *deep fields* [25] (cf. the color field in all three levels) for notational brevity.

Abstraction levels may include classification levels occupying different *modeling dimensions* [21]. See Figure 3 in [24], for instance, for how color coding segments and supporting layouting could meaningfully separate not only different classification levels, but also different classification dimensions, resulting in a model with constituting parts that are easy to parse on their own.

Diversity. Element diversity is observed when elements at the same abstraction level are conceptually related but differ in one way or another. For instance, the various phone or car models offered by manufacturers constitute domain diversity. The latter is a source of considerable complexity for "Industry 4.0" models [11], with their abundance of functionally similar but slightly varied components. High diversity typically manifests in models as specialization hierarchies that feature numerous subtype siblings. See Figure 3 for an example of a *diversity segment*.

Variance. A domain element exhibits variance when it may occur in different configurations while maintaining its identity. For

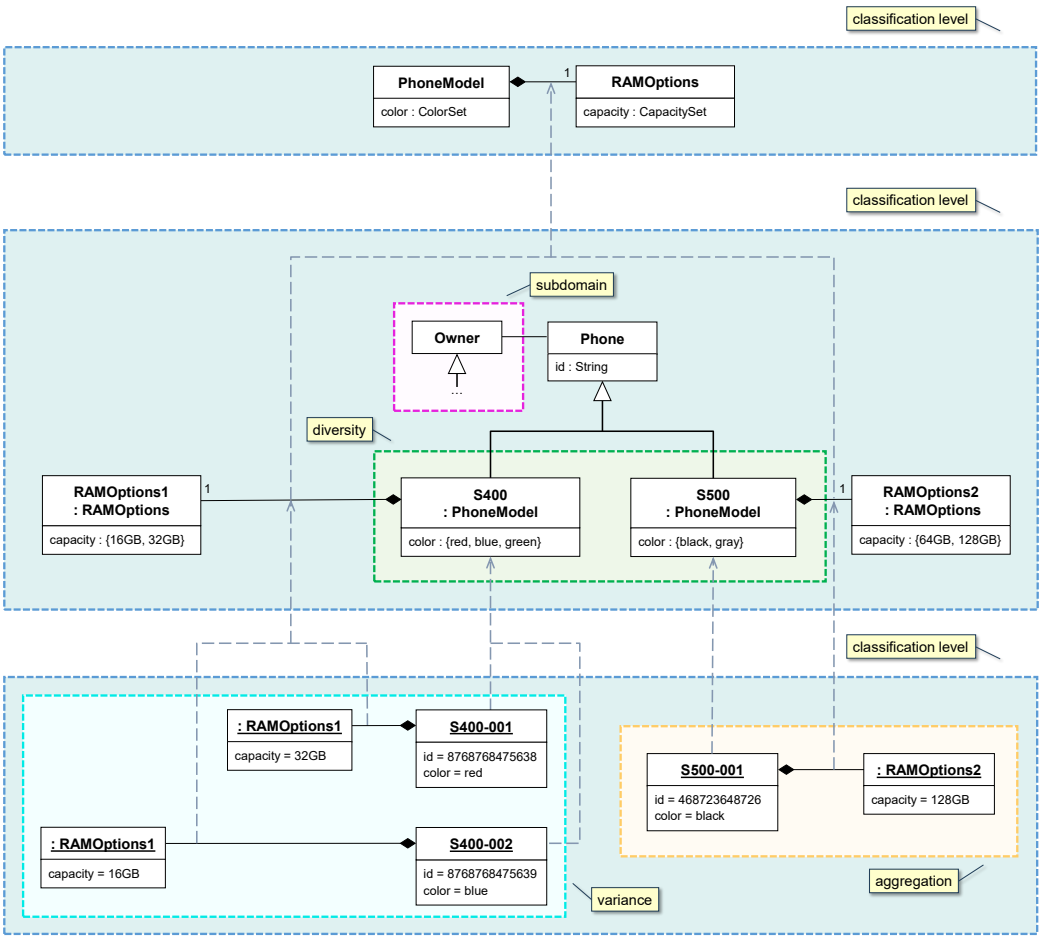

Figure 3: Segment Examples

instance, a particular car may be equipped with certain features, like heated seats, that may or may not be present.

In modeling, variance is often addressed via *feature modeling* [6], but it may just refer to the optionality of parts. In Figure 3, we identified two instances of phone model S400, which offers a certain set of configurations, as examples of variance (cf. the *variance segment*).

Subdomain. Complex domains can often usefully be divided up into thematically distinct subdomains. For example, in process modeling one may usefully separate activities from artifacts, or in manufacturing one may consider products separately from factories and customers respectively. Figure 3 shows the top of an elided Owner hierarchy, comprising Person, Company, etc., that can be considered to be a subdomain which is distinct to the "phone" subdomain referenced by the rest of the model elements in Figure 3.

In models, subdomains may explicitly manifest as packages, models that are part of a megamodel, or as dimensions in multi-dimensional MLM (MDMLM) [21]. Each subdomain may be represented with its own domain-specific modeling language.

Substructure. Complex domain scenarios often contain identifiable structures such as a *subsystem* or a *semantic fragment*. An example for the latter is the notion of a movie that requires multiple concepts to be comprehensively represented, including script, director, cast, etc.

We treat the notion of *substructure* as an umbrella term that should be further differentiated, e.g., into categories like semantic fragment, subsystem, etc., in order to capture modeler intent and further model element explainability.

3.2.2 Solution-Induced Sources. In contrast to the domain-induced segmentation sources listed in the previous section, solution-induced sources are motivated by the desire to structure designs or solution artifacts in general. They are hence not germane to domains and depend on solution choices, e.g., which technology or approach is employed. Segmentation sources solely driven by solution technology include:

Versions. Time does not only separate events on a timeline in the domain (cf. Section 3.2.1), it also gives rise to different model versions in the solution space. The latter are typically managed by using version control.

Design Patterns. Patterns in general, have been recognized in all phases of system development; design patterns specifically are used

to capture reoccurring best-practice solution mini-architectures [12]. Due to their structural regularity, it is possible to abstract away from their implementation details and replace them with named placeholders that reference the design pattern participants [13].

Aspects. It is oftentimes possible to identify so-called *concerns* in a model, e.g., score keeping and fuel modeling in a lunar lander game, that *cross-cut* through the chosen predominant decomposition strategy. Aspects provide a means to modularize such concerns into locally-defined descriptions that are woven into the, now aspect-free base model, whenever one wants to see the complete model (cf. "aspect-oriented programming" [18] and "aspect-oriented modeling" [19]). Similar to modules (see below), aspects can be represented using package-like structures and/or representatives that can be opened to reveal the internal structure.

Swimlanes. Sometimes behavior or structure is the Cartesian product of multiple contributing simple behavior/structures. In this case, one can factorize a complex description (the product) into multiple *swimlanes* (the factors), with the understanding that the latter need to be recombined in order to obtain the intended behavior/structure. For example, the UML uses swimlanes to factorize complex behavior into simpler sub-behavior for activity- and state-diagrams [28].

Modules. Compartmentalizing separable specification parts into encapsulated modules is a key strategy to decrease coupling in specifications, including models. The principle of a module, as it is directly supported in languages like Modula, may appear in various degrees of granularity and some segment-inducing sources, such as aspects or components, contribute to modularization. In the absence of more dedicated support, modules might manifest as packages in a module.

Stakeholders. Some approaches structure a model into parts according to the engineers/experts/modelers – in general *stakeholders* – that are responsible for these parts. Such a partitioning into stakeholder-induced segments may cross-cut through any other model language construct usages [15].

Other segmentation sources – while still typically being more relevant to a solution-oriented design phase rather than a problem-oriented analysis phase – may well be alignable with domain structure. These include:

Layers. A well-known example for using layers as a means to separate concepts at a different abstraction level is the OSI seven-layer model for system interconnection. The inter-layer relationship may be refinement and/or specialization.

Refinement. Many specification or modeling approaches, like the B method [30] or SySML, support *refinement* as a formal relationship that links an abstract concept with its concretizations. Depending on the definition of *refinement*, it may involve elements of instantiation and/or specialization, but may also just refer to notions of *completion* and/or *narrowing* that need not comply to regular instantiation/specialization.

Specificity. The difference between general-purpose languages and domain-specific languages is their level of *specificity* [3]. It is often possible to identify universal modeling concepts in models (cf. "upper ontologies") on the one hand, and domain/application-specific concepts on the other hand. *Specificity* may, but need not solely be, expressed using specialization. It may also be introduced by instantiating specific concepts from universal, or at least less-specific, types. Degrees of specificity can therefore often be found being distributed over multiple levels in an MLM model, providing a fine-structure to each level respectively.

Components. Similar to classes, components may correspond to respective domain entities, but they may also be purely solution-motivated. They form units of composition with explicit interfaces that provide behavior beyond the level of classes [31]. They are thus a prime candidate to be shown in a collapsed form that hides their internal structure, or in dedicated diagram types (cf. "component diagrams" in the UML).

3.3 Segment Representation

Model segments need to be represented in models either implicitly or explicitly. Some segment kinds straightforwardly map to existing model structuring mechanisms, e.g., domain classification levels directly map to classification levels in MLM models. Some segment kinds may be supported using specific language constructs, such as particular element relationships. For example, domain specialization typically maps to specialization relationships in an object-oriented model. Segment kinds without such explicit language support could be supported by dedicated segment-supporting language-structuring mechanisms. Failing all the above, it is always possible to let modelers manually tag elements to designate which model segment they should be considered to be a member of.

Table 1 provides an overview of typical segmentation support. Language support is contingent on the chosen modeling language while manually designation is always an option. However, in column "Manually Assigned" of Table 1 we only named manually tagged member elements, if manual management would not imply a loss of functionality. For instance, modules are predicated on some encapsulation support, hence merely tagging module members would not result in the normally desired effect.

The empty cells in column "Language Support" of Table 1 hint at potentially useful language support that, to the best of our knowledge, has not been explored yet. Regarding column "Head Types", we only included *organic head types*, in the sense that respective head elements are already part of the model. Of course, it is possible, for all segment kinds without a respective "head type" entry in Table 1, to let segments be represented by a dedicated segment representative, e.g., a named layer representative for a layer segment.

Figure 4 organizes the segment kinds listed in Table 1 (with the exception of the temporal *time* and *version* segment kinds) according to key properties: In addition to distinguishing between segments that have in-model heads and those who do not, we identify some segment kinds that lend themselves to be recursively nested (cf. Recursive Segment in Figure 4).

We furthermore recognize that some segment kinds can be grouped to form a cohesive entity (cf. Constituent Segment in Figure 4). Respective segments, e.g., levels that aggregate to a level hierarchy, have siblings, i.e., are related to elements in the same group. These segment kinds support efficient focusing since they allow all sibling segments to be temporarily hidden (see Section 4).

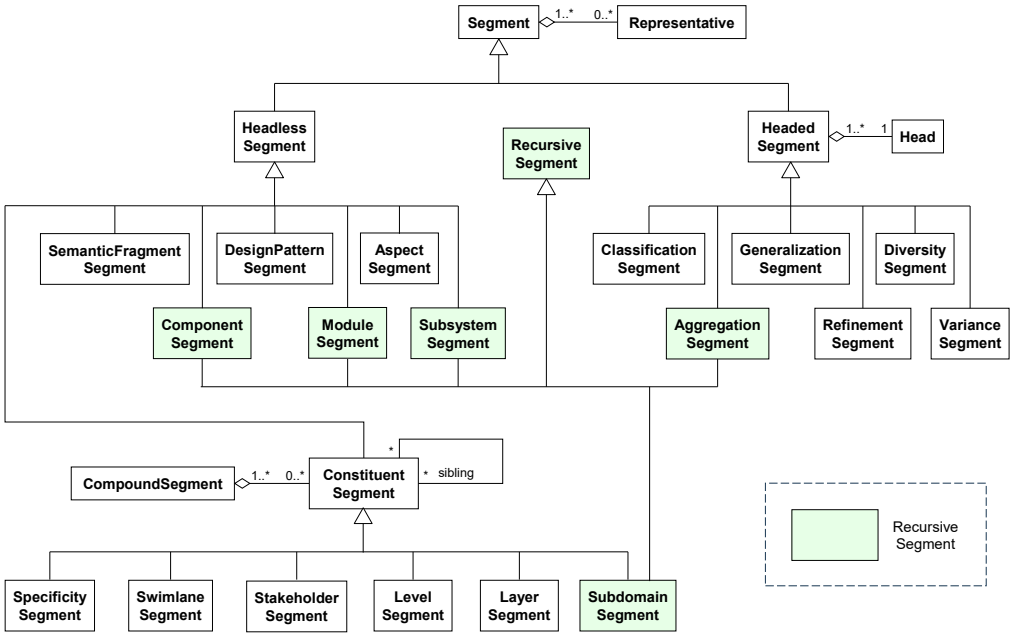

Figure 4: Segment Kinds

Note that Figure 4 is only meant to convey conceptual structure; it is not meant to detail a linguistic type model suitable for representing segments. Concepts like Representative and Head would have to be designed so that they can reference model elements.

Moreover, it may make sense to introduce dedicated variants of CompoundSegment, depending on which specific constituent segments are to be aggregated. Finally, there are a number of well-formedness constraints that could be expressed, which we do not cover here for brevity.

Finally, we do not claim Figure 4 is comprehensive. For instance, we neither included "manually assigned"-segment nor "modeling dimension"-segment. The former could be very flexible, e.g., allowing a head to be optionally chosen, or rather restrictive, and the details of the latter depend on particular assumptions which we do not want to commit to.

4 Segment Management

Independently from the segmentation sources and/or particular representation choices, various options exist to leverage the segmentation structure in a model to manage model complexity. In particular the notion of segment siblings, i.e., segments belonging to the same segment group, significantly enhances and extends traditional approaches to managing model complexity.

4.1 Spatial Separation

As illustrated by Figure 3, segments may be used to highlight model parts according to the segment kind they represent, thus helping to structure the model and provide hints to the model recipient as to the semantic role of a model part.

Segments are furthermore useful for both manual and automated layouting. Modelers may manually arrange whole segments to achieve an overall cleaner and more meaningful layout and tools may use segments as layouting hints.

As noted before, some segment kinds support hierarchical navigation (cf. Figure 4), i.e., allow entire model regions to be absent from the overall presentation, until their contents become relevant.

Since segments represent spatially localized entities, they also lend themselves to support model navigation. A model browser could support the targeting of individual segments by filtering for segment kinds, and or specific segments, e.g., via textual input such as *"Phone diversity"*.

4.2 Substitution

Segments open up the possibility to substitute a subset of the segment members or even the whole segment with a suitable representative. Options include

- replacing a subset of the members with one of them or a placeholder element.
- replacing all of the segment members with one of them or a placeholder element.
- replacing the entire segment with a proxy element.

For example, imagine reducing the diversity segment in Figure 3 to be replaced by a box that states "Phone models...". The aggregation segment in Figure 3 could be replaced by a single element S500-001 aggregate that acts as a proxy for the actual whole-part structure.

Representatives could be

- picked manually, via respective tool support.
- selected based on being "typical", e.g., based on a position within a calculated distribution, avoiding extreme values, lack of features, or degenerated elements in general.
- generated, e.g., by computing a prototype based on some combination strategy.

Segment Kind	Language Support	Manually Assigned	Head Type
time	potential support, e.g., via timestamps	time-aligned	world line
level	language-defined level		
classification	language-defined classification		type
generalization	language-defined generalization		generalization
aggregation	language-defined aggregation/composition		aggregate
diversity	language-defined one-to-many relationships		base
variance	feature modeling	template and variants	template
subdomain	MDMLM dimensions	subdomain members	
subsystem	language-defined subsystem	constituents	
semantic fragment		semantically related elements	
version	normally external to the language	version members	
design pattern	collaboration	pattern participants	
aspect	pointcuts	concern contributions	
swimlane	language-defined swimlane		
module	language-defined module		
stakeholder	package	stakeholder remits	
layer		layer members	
refinement	language-defined refinement/concretization		abstraction
specificity		equal-specificity elements	
component	language-defined component	component constituents	

Table 1: Segment Support

Such placeholders and proxies support so-called *chunking*, i.e., recoding model content with richer elements that condense detailed information using fewer symbols [9].

4.3 Temporal Filtering

Rathe than replacing elements or segments with placeholders one may also hide them. This will be even more effective for avoiding the effect of not being able to see the forest for the trees. For instance, it would be conducive to the purpose of understanding the phone variety design employed by the model in Figure 3, to entirely hide the variance segment. The latter does not add anything that cannot already be gathered by considering the aggregation segment. As a matter of fact, reducing the diversity segment to only show S500 would automatically remove segment variance and further simplify the model without taking anything away that would be needed to understand the phone variety design.

Figure 5 depicts how a modeler may focus on a specific part of the model by temporarily hiding all content in the segment siblings top-classification-level and bottom-level-classification level, and furthermore hiding a subdomain within the middle-classification level. Hidden parts are indicated by the use of a gray outline and ellipses in Figure 5. This scenario is an example of achieving focus by *isolating* an area of interest. Table 2 lists a number of operations that could be supported when segments are explicitly represented in models.

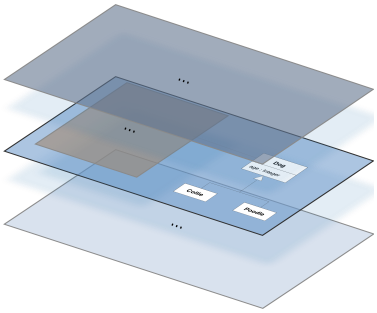

Figure 5: Segment Management

4.4 Customizable Views

It should be noted that segment operations, such as those suggested in Table 2, effectively allow modelers to create custom model views. In general, views are designed to project certain parts of a model and thus support focusing on aspects of interest, but do so in a fixed manner [4]. Dynamically configuring segment visibility is akin to creating a certain type of views on the fly.

As mentioned before, a segment navigator, as part of a model browser, could enable modelers to jump to a relevant part in the model and then hide other segments on a per-need basis. Such support would significantly expand the set of typically available navigation anchors, such as packages, classes, etc.

4.5 Collaboration Management

A useful side effect of a segmented model is that one may use the segments to *lock* certain segments for exclusive editing to support collaborative development, similar in spirit to the "*Lock and Commit Concept*" offered by Vector's PREEvision development tool [14]. Concurrent non-interference editing is supported by all segment kinds and one may even introduce a dedicated segment kind that is soley meant to delineate a lockable region (cf. Section 3.2.2).

Mode	Effect
isolate	selects segment and hides all other segments
focus	selects segment and hides all sibling segments
hide	removes segment from view
fold	replace segment with a proxy
show	brings hidden segment into view
unfold	replace proxy with segment

Table 2: Segment Operations

5 Multi-Level Modeling and Segmentation

While model segmentation is applicable to two-level and multi-level technologies alike, its application to MLM is particularly interesting. MLM's contribution to reducing model complexity historically has been characterized as addressing accidental complexity only. In this paper, we pointed out that MLM also helps managing essential complexity by allowing the domain segmentation source "classification level" to be mapped to segments in the form of *levels*.

Furthermore, the thus implied level hierarchy results in interactions with other segmentation sources. For instance, specificity segments are necessarily distributed to multiple classification levels [2, 3], whereas others like diversity segments are confined to single classification segments. A comprehensive systematic exploration into the topology of segment kind intersections remains future work, as is the extension of modeling languages with constructs that acknowledge domain-induced segments. The existence of certain modeling elements, e.g., subtypes in a generalization hierarchy, would then become explainable, in the sense of allowing the recipient of a model to understand modeling choices. For instance, the use of specialization could then be understood as arising from domain diversity or be the result of providing different solution-oriented realization strategies.

Two observations are particularly pertinent regarding adding segmentation to an MLM language: First, MLM may contribute to model complexity by allowing many hitherto unconnected models to be understood as variations of each other and thus bring them together in one model. Arguably, however, this amounts to an increase of single model complexity while the overall complexity of the totality of all (previously separated models) is not increased. Be that as it may, even for the increased single-model complexity, the use of subdomain or variance segmentation is a way to manage complexity by focusing at one previously isolated model at a time, thus effectively allowing the original level of individual complexity to be recreated on demand.

Second, whether deservedly or not, MLM has been associated with increased complexity. Despite the argument that MLM untangles bottom-level complexity—created by employing two-level technologies to represent multi-level domains—into explicit, language-supported modeling levels, the perception remains that MLM is a more complex technology, giving rise to more complex models.

Moreover, while it could be argued that the additional notation required for MLM models to support ideas like deep characterization is far offset by the respective gains, it remains the case that MLM notation is richer than most two-level modeling notations. Employing segments in MLM would allow MLM models to look more approachable due to the ability to explicitly recognize model substructures, introduce explainability, and replace detailed structures with semantic proxies. As a result, the model content that is shared with two-level technologies—within the level that contains order-1 concepts—would appear to be simpler and easier to manage in a segment-supported MLM model, compared to the respective content in a two-level model without segment support. We therefore maintain that enriching MLM languages with segment support would create a powerful antidote to the aforementioned issues.

6 Conclusion

Essential model complexity cannot be eliminated but can only be managed, e.g., with approaches like decomposition, abstraction, layouting, element elision, and the use of views. Decomposition has limits because there are element clusters that cannot be divided up without paying an untenable price for depriving modelers from seeing them in combination and for expressing how the parts would have to be composed. Abstraction can be useful to provide higher-level views on more detailed parts of the model, however, the respective elements add to the overall complexity, especially in case it is not self-evident that they are meant to provide a condensed view on more detailed parts. In the absence of view support, model layout is typically required to do the heavy lifting of making model content accessible. There are limits, though, as to how well a strategic 2D-arrangement of model elements can tame the complexity of substructures and relationship networks. Moreover, only a few model substructures are amenable to be treated as containers that can hide their inner structure. As a result, several modeling tools support element elision, but limit this support to elements and structures that are part of the modeling language.

In this paper, we proposed a way of elevating the aforementioned elision practices, to hide entire model segments and/or use the latter to yield both improved model navigation and more structured model presentation. We identified model segments as being representable by regular language constructs, dedicated segment representations, and manual user-designation. We argued that model segments can be implied by the domain but may also be motivated by solution structures. In other words, we argued the case for both analysis and design segments, both of which need to be represented within models, by leveraging existing constructs of the modeling language, and/or by extending the latter to support segments, based on notions like diversity or subdomain, to name just two examples.

We argued that model segments support a simplified view on models without adding to overall model complexity, as adding abstraction levels/layers to the model would. The respective simplification is not only achieved by supporting focus through dynamic navigation and filtering, but also by using segments in the presentation of models. As a matter of fact, segments not only break down big models into more manageable parts, but they also enrich model semantics by documenting the underlying cause for certain model structures. For instance, the plurality of elements caused by *domain diversity*, or *domain variance* can be explicitly recognized. Implied but hitherto invisible structures, such as those emerging from acknowledging different levels of specificity within the model, can be effectively communicated, using particular presentation strategies, e.g., involving segment labels and/or coloring schemes.

In contrast to previously recognized targets for model substructures, model segments occurring within coherent groups give rise to the notion of *segment siblings*. For instance, the classification levels in an MLM model are siblings and thus support hiding all levels except the one in focus. Color coding can communicate membership to the overall group (e.g., by using the same hue), while spatial separation and/or the use of different luminance values can be employed to differentiate siblings from each other.

The group character of segments is also crucial for supporting *chunking*, i.e., replacing a plurality of elements with proxies. The respective removal of proverbial "trees" significantly allows one to recognize the "forest", i.e., important model structures that are otherwise hidden in the noise of a myriad of model elements. This is important for models featuring overwhelming diversity, such as "Industry 4.0" applications. Previous model complexity management techniques only addressed container-like substructures, like packages, subsystems, and substates, while our generalization of such substructures to both solution- and domain-induced model segments can also manage diversity, variance, specificity, etc.

Furthermore, since many, in particular domain-induced, segments inject explicit structure into models that was previously not represented, they enable the efficient location of relevant model parts that otherwise could not be identified explicitly. A segment-supporting tool could not only allow modelers to browse concepts such as packages and classes, but also semantic entities like *"diversity sources for phones"* or *"whole-part structures for phone instances."*

We effectively extended the notion of using segregation and cohesion principles that have been used to leverage solution-based structures like layers, aspects, and swimlanes, to a much wider class of solution-based structures and even extended it to domain-induced structures. We have identified a wide range of segment kinds that far exceeds the kinds currently supported by tools for model substructure management, meaning that there is currently a significant untapped potential for improving complexity management by explicitly supporting further segment kinds.

Finally, we argued that segmentation gels very well with MLM, not only because MLM levels achieve a specific kind of segmentation already. Rather, the perception of MLM leading to more complex models can be counteracted by adding segmentation to MLM languages. We therefore hope that the contributions in this paper will prove to be useful in removing barriers to the more widespread adoption of MLM.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive feedback; in particular for pointing out the potential of using segments as lockable units to support non-interference editing.

References

[1] Colin Atkinson, Ralph Gerbig, and Thomas Kühne. 2015. A unifying approach to connections for multi-level modeling. In *Proceedings of MODELS'15*. IEEE Computer Society, NY, USA, 216–225. https://doi.org/10.1109/MODELS.2015.7338252

[2] Colin Atkinson and Thomas Kühne. 2000. Strict Profiles: Why and How. In *Proceedings of the 3rd International Conference on the UML 2000, York, UK*, Andy Evans, Stuart Kent, and Bran Selic (Eds.). Springer Verlag, LNCS 1939, 309–323. https://doi.org/3-540-40011-7_22

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

[4] Colin Atkinson and Christian Vjekoslav Tunjic. 2021. A Deep View-Point Language and Framework for Projective Modeling. *Information Systems* 101 (2021). https://doi.org/10.1016/j.is.2019.101440

[5] Frederick P. Brooks. 1987. No silver bullet: essence and accidents of software engineering. *Computer* 20, 4 (1987), 10–19. https://doi.org/10.1109/MC.1987.1663532

[6] Kacper Bąk, Zinovy Diskin, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. 2016. Clafer: unifying class and feature modeling. *Software and Systems Modeling* 15, 3 (July 2016), 811–845. https://doi.org/10.1007/s10270-014-0441-1

[7] S.N. Cant, D.R. Jeffery, and B. Henderson-Sellers. 1995. A conceptual model of cognitive complexity of elements of the programming process. *Information and Software Technology* 37, 7 (1995), 351–362. https://doi.org/10.1016/0950-5849(95)91491-H

[8] B. Combemale, J. Kienzle, G. Mussbacher, O. Barais, E. Bousse, W. Cazzola, P. Collet, T. Degueule, R. Heinrich, J. Jézéquel, M. Leduc, T. Mayerhofer, S. Mosser, M. Schöttle, M. Strittmatter, and A. Wortmann. 2018. Concern-Oriented Language Development (COLD): Fostering Reuse in Language Engineering. *Computer Languages, Systems & Structures* 54 (2018), 139–155. https://doi.org/10.1016/j.cl.2018.05.004

[9] John S. Davis. 1984. Chunks: A basis for complexity measurement. *Information Processing & Management* 20, 1 (1984), 119–127. https://doi.org/10.1016/0306-4573(84)90043-8 Special Issue Empirical Foundations of Information and Software Science.

[10] Juan de Lara, Esther Guerra, and Paolo Bottoni. 2025. Modular Language Product Lines: Concept, Tool and Analysis. *Software and Systems Modeling* 24 (2025), 981–1010. https://doi.org/10.1007/s10270-024-01179-9

[11] D. Galar Pascual, P. Daponte, and U. Kumar. 2019. *Handbook of Industry 4.0 and SMART Systems (1st ed.).* CRC Press, Florida, USA. https://doi.org/10.1201/9780429455759

[12] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Boston, MA.

[13] Martin Girschick, Thomas Kühne, and Felix Klar. 2006. Generating Systems from Multiple Levels of Abstraction. In *Trends in Enterprise Application Architecture: 2nd International Conference, TEAA 2006, Berlin, Germany, November 29 - December 1, 2006, Revised Selected Papers* (Berlin, Germany). Springer-Verlag, Berlin, Heidelberg, 127–141. https://doi.org/10.1007/978-3-540-75912-6_10

[14] Vector Informatik GmbH. 2026. Cross-Location Collaboration With PREEvision. Webpage. https://www.vector.com/int/en/products/products-a-z/software/preevision/collaboration-platform/#c373433 "Lock and Commit Concept", Last accessed: 10 August 2026.

[15] Cesar Gonzalez-Perez and Brian Henderson-Sellers. 2006. A powertype-based metamodelling framework. *Software and Systems Modeling* 5, 1 (April 2006), 72–90. https://doi.org/10.1007/s10270-005-0099-9

[16] Robert Heinrich, Misha Strittmatter, and Ralf Reussner. 2021. A Layered Reference Architecture for Metamodels to Tailor Quality Modeling and Analysis. *IEEE Transactions on Software Engineering* 47, 4 (2021), 775–800. https://doi.org/10.1109/TSE.2019.2903797

[17] Dazhou Kang, Baowen Xu, Jianjiang Lu, and W. C. Chi. 2004. A Complexity Measure for Ontology Based on UML. In *10th IEEE International Workshop on Future Trends of Distributed Computing Systems.* IEEE, New Jersey, 222–228. https://doi.org/10.1109/FTDCS.2004.1316620

[18] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-oriented programming. In *ECOOP'97 — Object-Oriented Programming*, Mehmet Akşit and Satoshi Matsuoka (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 220–242. https://doi.org/10.1007/BFb0053381

[19] Jörg Kienzle, Wisam Al Abed, Franck Fleurey, Jean-Marc Jézéquel, and Jacques Klein. 2010. *Aspect-Oriented Design with Reusable Aspect Models.* Springer, Heidelberg, 272–320. https://doi.org/10.1007/978-3-642-16086-8_8

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

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

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

[23] Thomas Kühne and Arne Lange. 2020. Meaningful metrics for multi-level modelling. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (Virtual Event, Canada) *(MODELS '20)*. Association for Computing Machinery, New York, NY, USA, Article 85, 9 pages. https://doi.org/10.1145/3417990.3421412

[24] Thomas Kühne and Pierre Maier. 2024. FMMLx and DLM – A Contribution to the MULTI Collaborative Comparison Challenge. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems* (Linz, Austria) *(MODELS Companion '24)*. Association for Computing Machinery, New York, NY, USA, 800–809. https://doi.org/10.1145/3652620.3688212

[25] Thomas Kühne, João Paulo A. Almeida, Colin Atkinson, Manfred A. Jeusfeld, and Gergely Mezei. 2023. Field Types for Deep Characterization in Multi-Level Modeling. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, New Jersey, 639–648. https://doi.org/10.1109/MODELS-C59198.2023.00105

[26] Ma Esperanza Manso, Marcela Genero, and Mario Piattini. 2003. No-Redundant Metrics for UML Class Diagram Structual Complexity. In *Advanced Information Systems Engineering: 15th International Conference. CAiSE 2003 Proceedings*, Johann Eder and Michele Missikoff (Eds.). Springer, Berlin and Heidelberg, 127–142. https://doi.org/10.1007/3-540-45017-3_11

[27] Gergely Mezei, Thomas Kühne, Victorio A. Carvalho, and Bernd Neumayr. 2021. The MULTI Collaborative Comparison Challenge. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 495–496. https://doi.org/10.1109/MODELS-C53483.2021.00077

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

[29] Jérôme Pfeiffer and Andreas Wortmann. 2026. A Theory of Composable Modeling Language Components. *Software and Systems Modeling* (2026). https://doi.org/10.1007/s10270-026-01382-w

[30] Emil Sekerinski and Kaisa Sere (Eds.). 1999. *Program Development by Refinement: Case Studies Using the B Method.* Springer, Berlin and Heidelberg.

[31] Clemens Szyperski. 1999. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley, Boston, MA.

[32] David Ungar and Randall B. Smith. 1987. Self: The power of simplicity. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM press, USA, 227–242. https://doi.org/10.1145/38765.38828

[33] Horst Zuse. 1991. *Software Complexity: Measures and Methods*. Walter de Gruyter, Berlin and New York.