Property-Precedence Analysis for Model Deepening

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

Abstract

Despite widely accepted prospects, multi-level modeling remains sparsely adopted in industry. *Model deepening* –, i.e., the re-engineering of flat representations into multi-level models – has the potential to facilitate the industrial adoption of multi-level modeling by transforming models and representations of information systems an organization already possesses. Prior research has focused primarily on transformation operations; comparatively little attention has been devoted to the preceding analysis problem of identifying beneficial model-deepening transformations. Existing approaches typically rely on type-level heuristics such as naming patterns, aggregation relationships, or association multiplicities. Since these heuristics suggest multi-level hierarchies based on modeling choices and conventions rather than the represented instance population, they cannot reliably distinguish genuine classification levels from incidental modeling choices.

This paper proposes *property-precedence analysis*, an instance-driven approach to conduct model-deepening analysis. Building upon Bunge's ontological notion of property precedence, the proposed approach uncovers possible multi-level hierarchies by analyzing the distribution of property values across model instances rather than relying on type-level heuristics. The analysis is implemented in the prototype ModelDeepener© for the MLM language FMML^{x} and demonstrated using two example models involving associations and attributes. Several challenges in property-precedence analysis for model deepening remain, such as handling conflicting precedence relations and naming newly suggested classes. Further research should investigate to what degree the identified challenges may be counteracted and how property-precedence analysis may be complemented with type-level heuristics for model deepening.

CCS Concepts

• **Software and its engineering → Object oriented development**; **Abstraction, modeling and modularity**.

Keywords

model deepening, property precedence, multi-level modeling

ACM Reference Format:
Pierre Maier. 2026. Property-Precedence Analysis for Model Deepening. 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/10.1145/3837062.3838704

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/10.1145/3837062.3838704

1 Introduction

The prospects and benefits of multi-level modeling (MLM) over traditional modeling and language-engineering approaches have been discussed for over 20 years [2, 10]. Still, MLM remains a niche research topic and does not receive the attention it seems to deserve. Several factors contribute to this unsatisfactory circumstance. First, the current landscape of MLM approaches is characterized by a broad heterogeneity. MLM languages interpret and implement core language constructs differently, such as the notion of 'level' itself [16]. Despite various unification efforts [1, 15, 22], no standard MLM language has yet been developed. For organizations trying to adopt MLM, this heterogeneity presents a threat to their investment. Second, expertise in MLM is sparse. Apart from the modest number of researchers actively involved in MLM, it can also be assumed that MLM is sparsely taught in universities. It is not mentioned as an optional or mandatory competency area in the most recent curricular recommendations for Computer Science published by the ACM, IEEE, and AAAI [19]. Third, given that organizations have accumulated a mass of flat models, i.e., models that comprise at most two modeling levels, realizing the benefits of MLM would impose a massive, error-prone re-engineering effort. This applies especially in cases where even dedicated flat, two-level models are missing and, e.g., only software code or database files are available.

Against this background, it is relevant to investigate whether at all, to what degree, and under what conditions flat models can be redeveloped as multi-level models. I refer to the process of re-engineering flat models into deep multi-level models as model deepening [23]. This re-engineering process should be automated as much as possible in order to increase the efficiency and quality of model deepening. Model deepening consists of at least two phases: an analysis phase and a transformation phase. In the analysis phase, an input flat model is analyzed in order to determine which elements of the model may be 'deepended,' i.e., changed in order to benefit from applying MLM-specific features. In the transformation phase, a sequence of change operations is applied to the flat model as suggested by the analysis phase. While these two activities are closely related, they address fundamentally different problems: the former concerns a contingent semantic analysis, whereas the latter concerns deterministic change operations.

While some publications discuss change operations for MLM in general [6, 28, 29], few address model deepening. And publications discussing how to re-engineer flat models into multi-level models focus almost exclusively on the required change operations to rearchitect the type-object pattern, sidestepping issues of detecting deepening potentials in flat models (see also [3]). Macías et al. [21, 22] present the so-called *MLM-Rearchitecter* tool. The tool rearchitects annotated type-object patterns into multi-level models but does not put much emphasis on the detection of those patterns,

as noted by Macías et al. [21, p. 63]: "Several heuristics are possible to automatically detect occurrences of the type-object pattern, e.g., based on naming conventions [...]. However, at this stage, we have focussed on [*sic*] the translation of different variations of this pattern into a MLM setting, leaving pattern detection heuristics for future work." The most recent version of the tool[1], which does not seem to have been updated since the original publications, does not go beyond a rule-based pattern matching of class names in order to detect type-object patterns. Part of the name pattern matching is to analyze whether a class name has the suffix 'Type.' Neumayr and Schrefl [26] assume that domain hierarchies in flat models are modeled via aggregation relationships: "The set of classes (and hence levels) in a domain object hierarchy is arranged [...] by aggregation relationships" [26, p. 589]. Accordingly, they suggest to re-engineer all aggregation relationships as classification relationships. In a more recent publication, I suggested analyzing the multiplicities of associations in order to detect type-object patterns [23]. The idea behind analyzing the multiplicities of associations is that some associations may intend to emulate classification relationships. Since in class-based object-oriented languages an object must be a direct instance of exactly one class (multiplicity 1..1) and each class may have an arbitrary number of instances (multiplicity 0..*), associations replicating this multiplicity may be re-engineered as classification relationships.

Current model-deepening analysis approaches thus predominantly rely on heuristics observable at the type level. All of these heuristics are based on simplifying assumptions that are faced with two core issues. (1) First, they do not support distinguishing between false positives and true positives. A class may include the suffix 'Type,' may be related to another class via an aggregation relationship, or may have an association imitating the classification multiplicity and still not serve to instantiate objects that serve as types for other objects. Rearchitecting false positives into multi-level hierarchies is not suited to realize the benefits of MLM. *Example:* Assume we have two classes `Address` and `Customer` related by an association with the classification-imitating multiplicity (a person has exactly one address, each address may belong to none or many customers), making it technically possible to re-engineer it as a classification relationship by promoting the supposed 'type class' (`Address`) to level 2. Doing so would not only make the model confusing (why should `Customer` be considered an instance of `Address`?), it would also threaten the integrity of the model if, at some time, the multiplicity between customers and address may change, allowing, e.g., a customer to possess multiple addresses. Then, the classification hierarchy is erroneous and must be refactored along all side effects that come from changing the classification hierarchy. (2) Second, the heuristics similarly do not support distinguishing between true negatives and false negatives. Just because a certain heuristic is not followed, does not mean that the represented elements may not be better arranged in a multi-level hierarchy. Modelers may follow different naming conventions, not using the suffix 'Type' to mark type classes, they may avoid using aggregation relationships, or they may specify association multiplicities less restrictively. Each of these heuristics demands that a user follows a convention they may not even be aware of. In any case, a lack of detecting further model snippets well suited for model deepening fails to realize the full potential of MLM.

Given these issues with current analysis approaches for model deepening, this paper proposes a complementary approach to analyze flat models for model deepening called *property-precedence analysis*. Property-precedence analysis is an instance-driven model-deepening approach. The central assumption is that instance-level data may implicitly contain structural information that is often not explicitly expressed at the type level.

The objective of this paper is, however, not to present a complete implementation and evaluation of this analysis approach. Given the problems faced by alternative heuristics-based approaches, the purpose is to outline an alternative analysis approach, demonstrate its feasibility, and outline core challenges. For the purpose of this paper, a number of simplifying assumptions are made. First, not all model properties are analyzed. Operations and constraints, being also not supported by every MLM language, are not considered within the scope of this paper. Second, attribute multiplicity is not taken into account. All regular class attributes are assumed to have a multiplicity of 0..1 or 1..1. Third, while the idea of property-precedence analysis is outlined in an MLM-language-agnostic way, a number of experiments have been conducted with a model-deepening tool, called ModelDeepener©[2] that is tailored towards FMML^x models. The Flexible Multi-Level Modeling and Execution Language (FMML^x) is an object-oriented MLM language. Since the constructs offered by FMML^x are different to other object-oriented MLM languages (see [18, 20] for a comparison of other MLM languages to FMML^x), the implementation illustrated here cannot be applied without adaptions to other MLM languages.

In Sec. 2, the core terminology of model deepening is outlined, accompanied by a concise description of specific features of FMML^x. In Sec. 3, the property-precedence analysis is outlined with an example FMML^x model. In Sec. 4, the core functions of the property-precedence analysis, as currently implemented in ModelDeepener©, are described. In Sec. 5, two example cases of property-precedence analysis are presented. The insights gathered from the prototypical development and demonstration, as well as remaining challenges, are discussed in Sec. 6. A conclusion is provided in Sec. 7.

2 Model Deepening and FMML^x

2.1 Overview of General Functions in Model Deepening

Model deepening denotes the process of re-engineering flat models into multi-level models. Flat models comprise at most two modeling levels. A deep multi-level model is characterized by at least one of the following MLM-specific features. First, the model spans across multiple modeling levels. Second, the model includes elements that posses an instance and a type facet. Third, the instantiation of some properties is deferred, i.e., they are not instantiated on the level immediately below. Independent from instantiation semantics, the latter feature is sometimes referred to as *deep characterization*. These three MLM-specific features are assumed to constitute the common core of MLM languages [15]. Specific MLM languages

[1] https://github.com/MLM-Rearchitect-Exchange/mlm-rearchitecter-tool, last accessed 23 June 2026

[2] https://github.com/model-deepening-research/Model-Deepener, last accessed 06 August 2026

may also offer additional features. Note that just because a model is created with an MLM language does not imply necessarily that it is also deep. Within this paper, I consider the expression 'multi-level model' to refer to deep models only.

The term model deepening in the sense presented here was first used in [23]. The name has been introduced for two reasons. First, it mirrors the expression 'model flattening' used, e.g., by Guidoni et al. [14] to denote abstraction-removing transformations. Second, it resembles the notion 'deep modeling,' which is sometimes used to refer to potency-based MLM approaches [7]. The notion of 'rearchitecting' two-level models into multi-levels used by others [3, 21], which is commonly mentioned in this context but lacks a precise definition, is here taken to encompass the applied change operations in model deepening that re-organize elements of the original model.

Model deepening is characterized by three core functions (see Fig. 1). Function 1 ("perform exogenous model transformation") receives a flat model expressed in some conventional, non-MLM language like UML. This flat model is then transformed into the target MLM language. The model remains flat, i.e., restricted to two modeling levels; it is just expressed with an MLM language (see [24] on how to express UML-like class and object diagrams with FMML^{x}). This function is not further discussed within this paper. Function 2 ("perform model-deepening analysis") receives as input a flat model in the target MLM language either from a user or from function 1. The received flat model is analyzed in order to determine how it may be changed into a multi-level model. While a variety of analysis approaches may be distinguished here, note that within this paper, only the so-called property-precedence analysis is considered. Based on the results of the model-deepening analysis, the required change operations are performed in order to rearchitect the flat model into a deep multi-level model (function 3, "apply change operations to model"). This function corresponds to an endogenous model transformation. The transformed model may be again used for a further analysis of model-deepening potentials. Note that Fig. 1 represents a data-flow diagram and as such does not represent control flows in model deepening.

Model deepening should not be considered purely syntactic. Multi-level models make conceptual distinctions explicit that remain implicit in flat models, requiring semantic information to be reconstructed during the transformation. Consequently, model deepening should be understood as a re-engineering and not a transformation process; it involves a semantic reconstruction of available information rather than a mere rearchitecting of existing models. Since multi-level models are semantically more expressive than flat models, the process necessarily involves an inductive leap. The inductive leap in model deepening is illustrated in Sec. 3.2.

2.2 Concise Description of FMML^{x}

The FMML^{x} is an object-oriented and order-synchronized [16] MLM language. The meta model of the FMML^{x} is based on a golden-braid language architecture [8, 12] and not on the more common orthogonal classification architecture. Thus, FMML^{x} does not distinguish between so-called linguistic and ontological classifications.

In FMML^{x}, everything is an object. Some objects may possess a class facet, too, which enables them to specify type properties (association ends, attributes, operations, constraints) and provides them with the capability to instantiate further objects that themselves may possess a class facet yet again. Each object is assigned an integer level value. The level of objects in FMML^{x} expresses the level membership and the possible concretization/instantiation depth of the respective object. Other MLM languages use two separate integer values for this purpose: a level value to indicate level membership and a so-called potency value that serves to reflect the possible instantiation depth. Whether or not objects possess a class facet in FMML^{x} depends upon their level. L0[3] objects never possess a class facet and can thus never be instantiated any further. Any object on a level greater than zero possesses a class facet.

Each class property is assigned an instantiation level that must be at least one level below the level of the owning class. The instantiation level determines at which level the property is to be instantiated. If the instantiation of a class property is deferred, i.e., the instantiation level of a property is not exactly one below the level of its owning class, the property is inherited to lower-level descendants of the owing class until the instantiated object's level matches the instantiation level of the property. Thus, FMML^{x} does not implement a pure classification/instantiation relationship between classes on neighboring levels. The combination of instantiation and inheritance that occurs between classes on neighboring levels is called concretization/intrinsic classification [11]. Note that since L0 objects may not possess a class facet, no inheritance may occur between L1 classes and L0 objects. The relationship between L0 objects and L1 classes therefore corresponds to the traditional instance-of relationship. FMML^{x} currently provides support only for single-dual deep fields [17]: values for attributes can only be provided when an attribute is instantiated as a slot. The instantiation level of class properties in FMML^{x} is comparable to the feature potency of other MLM languages. But unlike feature potency, which is based on a relative level-targeting mechanism specifying the number of instantiation steps until a property is to be instantiated, the instantiation level is based on an absolute level-targeting mechanism. Note that FMML^{x} also provides limited support for relative level targeting in order to support a specific kind of FMML^{x} classes called contingent-level classes [11]. This special feature is, however, not further discussed within the paper. FMML^{x} allows for level-crossing associations and links. That is, an association may connect two classes on different levels and each association end may specify a different instantiation level.

An example FMML^{x} model illustrating the outlined features is displayed in 2. The notation of FMML^{x} diagrams resembles standard UML class and object diagrams. The level of the object is displayed in the top-most compartment and the instantiation level of class properties are displayed next to a property. FMML^{x} also features a color-specific coding of objects according to their respective level.

FMML^{x} models are executable and support the specification of operations and constraints with the multi-level programming language XOCL (executable object command language) [5]. FMML^{x} models can be created and executed during run time with the dedicated modeling and language-engineering tool XModeler^{ML} [25].

[3]Levels in FMML^{x} are typically abbreviated with L<*level-number*> instead of the UML standard M<*level-number*>. This is due to the slightly different inter-level semantics in FMML^{x}. The level abbreviation *L* should not be confused as a 'linguistic' classification level.

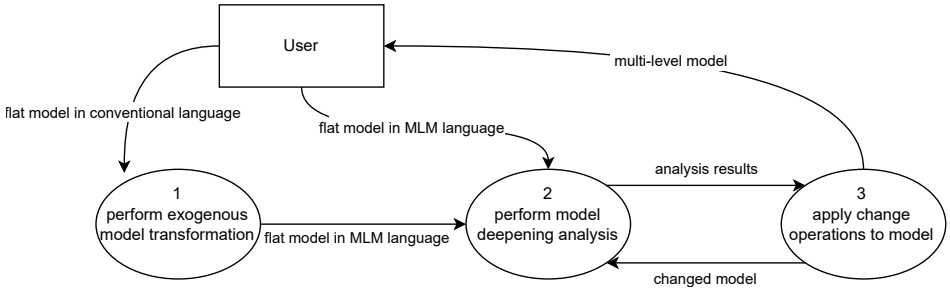

Figure 1: Data-flow diagram of core functions in model deepening

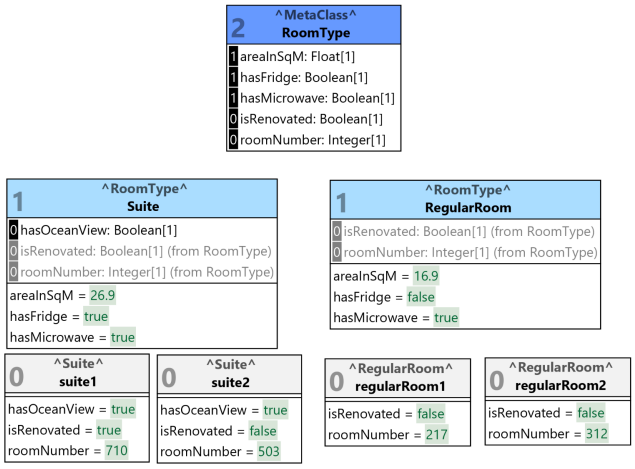

Figure 2: Example deep FMML^{x} model of hotel room types

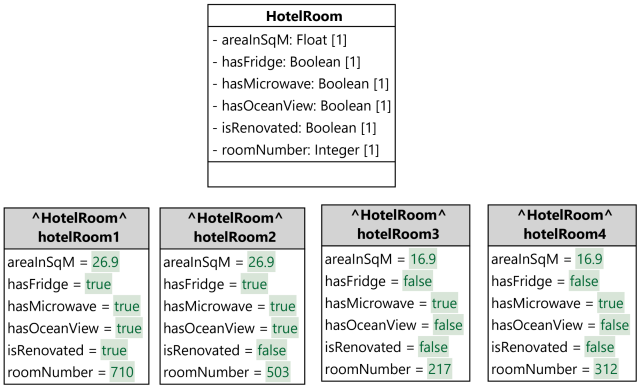

Figure 3: Possible flat counterpart of multi-level model from Fig. 2

Models created with XModeler^{ML} can be persisted as XML files. Further information on FMML^{x} and XModeler^{ML} are provided, inter alia, in [8, 25].

3 Outline of Property-Precedence Analysis

The previous sections establish that model deepening requires recovering conceptual distinctions that are implicit rather than explicit in flat models. Instead of analyzing the type-level characteristics of an input model, this paper proposes analyzing the information encoded in its instances. The underlying assumption is that instance-level data may contain semantic information relevant for MLM that are not explicitly accounted for on the type level of flat models. For this purpose, the precedence of model properties should be analyzed.

3.1 Property Precedence

Motivating Example. Classes and objects in conceptual models possess properties. Within the context of this paper, I focus on attributes and association ends as properties provided on the type level (*type properties*) and slots and slot links – slot links are considered the counterpart to association ends on the instance level – as properties provided on the instance level (*instance properties*).

The core assumption of this paper is that a multi-level model imposes an order of type and instance properties that is often not explicitly accounted for in flat conceptual models. Consider the example FMML^{x} model in Fig. 2. The particular area of different kinds of hotel rooms is represented by the slots for the attribute `areaInSqM`. Provided this multi-level model, it is clear that rooms have the same area based on the category they belong to: suites always have an area of 26.9 m², regular rooms always have an area of 16.9 m². The model allows to infer a number of relations between properties. Based on the multi-level model, we know that each room classified as a suite has a fridge and only suites may have an ocean view.

This information may be expressed by a variety of ways in a flat model, one possible solution is displayed in Fig. 3. Here, the room category 'suite' is not explicitly accounted for. The room area is specified redundantly for each specific suite. The explicit conceptual information present in the corresponding multi-level model is lost. It is not clear that all regular rooms have an area of 16.9m². It is also not stated that only suites can have an ocean view.

Bunge's Property Precedence. The relation between properties in a multi-level model may be described as a *property precedence* relation. Property precedence relations are introduced by Bunge [4] as part of this ontological discussions.[4] Bunge maintains that things

[4]It is important to note that Bunge's contributions are part of ontology in philosophy which denotes the field that studies being and non-being. This notion of ontology is different to the common use in Computer Science and Information Systems research, where an ontology is commonly conceived as "an explicit specification of a conceptualization" [13, p. 908] and is thus understood similarly to conceptual models.

possess properties and some properties may be more general than others. He notes, for example, that "the property of thinking follows that of being alive, which in turn follows that of containing genetic material" [4, p. 81]. He provides the following formal definition of property precedence.

Let $\mathcal{S}(P)$ denote the set of all objects possessing property P, referred to as the *scope* of the property. A property P_1 *precedes* another property P_2 ($P_1 \preccurlyeq P_2$) if every object possessing P_2 also possesses P_1, while the converse may not hold. Formally,

$$P_1 \preccurlyeq P_2 :\Leftrightarrow \mathcal{S}(P_2) \subseteq \mathcal{S}(P_1)$$

Bunge also suggests the expressions P_1 "is more common than" or "is necessary for" P_2 [4, p. 80]. In the reverse direction, P_2 "follows" or "is sufficient for" P_1 (ibid.). Two properties are furthermore defined as *concomitant* ($P_1 \sim P_2$) if the scopes of both properties are equal. That is,

$$P_1 \sim P_2 :\Leftrightarrow \mathcal{S}(P_1) = \mathcal{S}(P_2)$$

Property precedence as defined by Bunge is an ontological relation. To Bunge, it expresses a relation of properties that applies to actually existing things. This is in stark contrast to conceptual modeling, where classes and objects possess properties by definition, through the design of a modeler. The notion of property precedence as defined by Bunge has been applied to conceptual-modeling literature before. For example, Parsons and Wand [27] have applied property precedence to uncover integration potentials between different flat models.

3.2 Example Illustration of Property-Precedence Analysis

Based on a given population of instances, it may be possible to induce property-precedence relations in flat models on the basis of which higher-level classes can be introduced. Property-precedence analysis follows four distinct steps (see Fig. 4). In the following, each of the steps is illustrated referring to the attributes `roomNumber` and `areaInSqM` from Fig. 3.

Step 1–Create Slot Collectives. Each room has a different room number, whereas multiple rooms have the same area. For each unique value of these attributes, we can detect the following scopes (`hotelRoom` is abbreviated as hr):

- For instance values of the attribute `roomNumber` (rn):
  - $\mathcal{S}(rn = 710) = \{hr1\}$
  - $\mathcal{S}(rn = 503) = \{hr2\}$
  - $\mathcal{S}(rn = 217) = \{hr3\}$
  - $\mathcal{S}(rn = 312) = \{hr4\}$
- For instance values of the attribute `areaInSqM` (ar):
  - $\mathcal{S}(ar = 26.9) = \{hr1, hr2\}$
  - $\mathcal{S}(ar = 16.9) = \{hr3, hr4\}$

Each entry in the above list is considered a *slot collective*. Slot collectives are defined as collections of slots, where each slot within one slot collective has the same value. Slot collectives have a scope that corresponds to the objects possessing a slot with the value of the slot collective.

Step 2 – Infer Precedence of Slot Collectives. Following the definitions of Bunge, we may infer the following precedence relations between these slot values:

- $\mathcal{S}(rn = 710) \subset \mathcal{S}(ar = 26.9) :\Leftrightarrow ar = 26.9 \preccurlyeq rn = 710$
- $\mathcal{S}(rn = 503) \subset \mathcal{S}(ar = 26.9) :\Leftrightarrow ar = 26.9 \preccurlyeq rn = 503$
- $\mathcal{S}(rn = 217) \subset \mathcal{S}(ar = 16.9) :\Leftrightarrow ar = 16.9 \preccurlyeq rn = 217$
- $\mathcal{S}(rn = 312) \subset \mathcal{S}(ar = 16.9) :\Leftrightarrow ar = 16.9 \preccurlyeq rn = 312$

The detected precedence relations between slot collectives express, e.g., that each room object that has the room number 710 has an area of 26.9 m².

Step 3 – Infer Precedence of Attributes. The most important step in property-precedence analysis is to infer the possible precedence of attributes. Attribute precedence may be inferred by comparing all slot collectives of two attributes. The slot collectives of the attributes `areaInSqM` and `roomNumber` always result in the same precedence relations – all slot collectives of `areaInSqM` precede slot collectives of `roomNumber`. From this, we may infer the following precedence between both attributes:

$$areaInSqM \preccurlyeq roomNumber$$

Even though the inference is straightforward and unambiguous based on the provided instance data, it is important to note that the inference of precedence between these attributes performs an *inductive leap* Based on a set of observed instance values, we induce that this relation also occurs for any future objects containing the same value. Since this performs an inductive leap, attribute precedence may be subject to the inductive fallacy: further objects may invalidate the inferred precedence relation.

Step 4 – Create Property-Precedence Graphs. The induced attribute precedence can be represented as a directed, acyclic graph called *property-precedence graph*. Each attribute is represented by a node. If attribute P_1 precedes attribute P_2, a directed edge is added pointing from P_1 to P_2. In the example considered here, the property-precedence graph would only consist of two nodes with a directed edge from `roomNumber` to `roomAreaInSqM`. Examples of property-precedence graphs are discussed in Sec. 5. Having induced the precedences of properties of a flat model, it is possible to generate a candidate deep model solution. The simple precedence graph consisting only of two nodes can be interpreted as follows. Each room has a room number, the value of which is preceded by a specific area. For each identified room area, a conceptually distinct class can be introduced to, in this example, represent suites and regular rooms. Just like the deep model in Fig. 2, these two classes with specific room areas can be added as instances on level 1 of a meta class called `RoomType`.

4 Prototypical Implementation of Property-Precedence Analysis in ModelDeepener©

Based on the outline of property-precedence analysis in Sec. 3, this section introduces the prototypical implementation of property-precedence analysis as implemented in the ModelDeepener© tool. In Sec. 4.1, the representation of flat and deep models in the tool is described. In Sec. 4.2, details on the implementation of the property-precedence analysis are provided. A UML class diagram providing

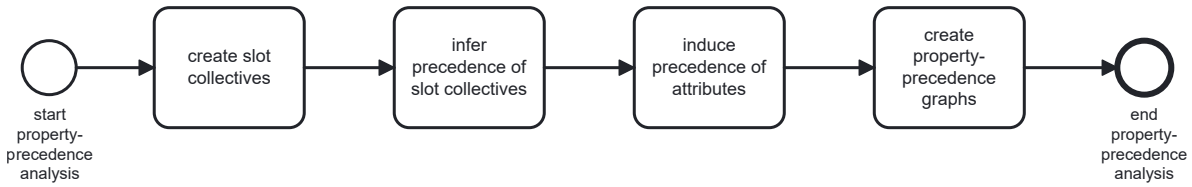

Figure 4: Simplified workflow of property-precedence analysis

an overview of the implemented classes in ModelDeepener© is provided in Fig. 5.

4.1 Representation of Flat and Deep Models in ModelDeepener©

The representation of models in ModelDeepener© is tailored towards FMML^{x} models. The program currently supports to import FMML^{x} models via XML files that have been exported using the XModeler^{ML}, or to import CSV files. Further import options, corresponding to function 1 in Fig. 1, can be added in the future. A flat or deep model is an instance of the class `FmmlxModel`. Each instance of this class includes a sets of objects, instances of the class `FmmlxObject`. As defined in FMML^{x}, each object included in a model has a name and a level integer value. Additional attributes to capture a potentially different potency or order values [16], as defined in other MLM languages, are not accounted for. Each object possesses a number of properties, among them attributes and slots. Each attribute has a name, a type, and an instantiation level; each slot a value and a link to the attribute it is an instance of. All slots, regardless of its attribute's actual type, are maintained as Strings. An FMML^{x} object may be an instance of a further FMML^{x} object.

Not displayed in Fig. 5 but part of the tool implementation are the classes `FmmlxAssociationEnd` and `FmmlxSlotLink`. Association ends are specializations of attributes. They account for associations between classes in a model. Just like regular attributes, association ends have a name, a type, and a multiplicity. The name of an association end does not correspond to the name of an association but instead to the role name/identifier provided for the association end in the original model. The type of an association end always corresponds to the class referenced by the original association. FMML^{x} also implements associations via association ends [12]. Slot links serve as the corresponding counterpart of association ends for slots. They are a specialization of `FmmlxSlot` and are linked to the respective association end they are an instance of.

The instances of all classes that are preceded with `Fmmlx` are created upon importing a model. An instance of `FmmlxModel` is considered a flat model if the level of all objects it contains is not higher than 1, else it is considered a deep, multi-level model.

4.2 Creation of Property-Precedence Graphs

As outlined in Sec. 3.2, the final objective of the property-precedence analysis is the creation of property-precedence graphs that serve as the foundation to infer candidate multi-level hierarchies. A property-precedence graph is a directed, acyclic graph. All nodes of this graph serve to represent a particular property of a model; a directed edge between two nodes $A \rightarrow B$ serve to represent the precedence of P_1 to P_2 (reading "P_1 precedes P_2").

Two kinds of property-precedence graphs are distinguished: *attribute-precedence graphs* and *slot-precedence graphs*. All nodes of an attribute-precedence graph must represent attributes and all nodes of a slot-precedence graph must represent slot collectives. Note that association ends are considered attributes and slot links are considered slots. For each object of a model, exactly one attribute-precedence graph and exactly one slot-precedence graph is created.

Step 1 – Create Slot Collectives. In order to create a property-precedence graph, slot collectives must be created first. A slot collective groups together all slots that contain the same value. The current implementation considers two slots to possess the same value if their String value is identical. It is possible to ignore the case of values. In order to create the slot collectives, the attributes of the class are iterated. For each instance of the class, which correspond to the L0 objects of a flat model, we retrieve the slot for the attribute. If a new slot value is encountered, a new slot collective is created and appended to the list of slot collectives of the flat class. If the slot value has been encountered before, we retrieve the already created slot collective of the value. In each case, the encountered slot is added the slot collective and the instance is added the scope of the slot collective.

Steps 2 and 3 – Infer Precedence of Slot Collectives and Attributes. After having created the slot collectives for each attribute of a class, the precedence relations between them can be inferred. All attributes of a class are compared to each other based on the slot collectives of each attribute. Since each slot collective has a scope, the identification of the relation between two slot collectives is straightforward. Consider two slot collectives S_1 and S_2 that each have a scope set $\mathcal{S}(S_1)$ and $\mathcal{S}(S_2)$ respectively. Then, three cases for comparing slot collectives can be distinguished:

(1) $\mathcal{S}(S_1) \subset \mathcal{S}(S_2)$: The scope of S_1 is a true subset of the scope of S_2
(2) $\mathcal{S}(S_2) \subset \mathcal{S}(S_1)$: The scope of S_2 is a true subset of the scope of S_1
(3) $\mathcal{S}(S_1) = \mathcal{S}(S_2)$: The scope of S_1 is equal to the scope of S_2
(4) $\mathcal{S}(S_1) \cap \mathcal{S}(S_2) = \emptyset$: The scopes of both slot collectives are disjoint

These comparisons of the scopes of the slot collectives indicates the following relation between them:

(1) $S_1 \prec S_2$ if $\mathcal{S}(S_2) \subset \mathcal{S}(S_1)$
(2) $S_2 \prec S_1$ if $\mathcal{S}(S_1) \subset \mathcal{S}(S_2)$
(3) $S_1 \sim S_2$ if $\mathcal{S}(S_1) = \mathcal{S}(S_2)$

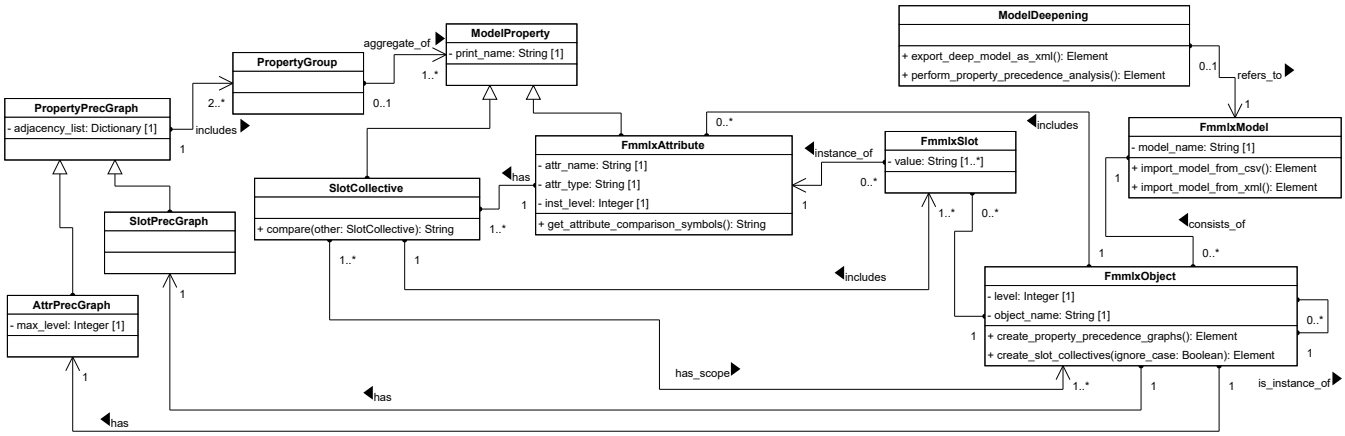

Figure 5: Excerpt of class structure for property-precedence analysis in ModelDeepener©

Two attributes A_1 and A_2 are then compared based on the results of the comparisons of all slot collectives of the attributes. This can lead to the following induced relations between both attributes, with Q and T representing the set of slot collectives for A_1 and A_2 respectively:

(1) $\forall q \in Q \subseteq A_1, t \in T \subseteq A_2 : s \sim t \implies A_1 \sim A_2$
(2) $\forall q \in Q \subseteq A_1, t \in T \subseteq A_2 : (s \prec t) \vee (s \sim t) \implies A_1 \preccurlyeq A_2$
(3) $\forall q \in Q \subseteq A_1, t \in T \subseteq A_2 : (t \prec s) \vee (s \sim t) \implies A_2 \preccurlyeq A_1$

The cases where two attributes A_1 and A_2 contain slot collectives the precedences of which are contradictory, i.e., some slot collectives of A_1 precedes some slot collectives of A_2, while also some slot collectives of A_2 precede some slot collectives of A_1, are not accounted for.

Step 4 – Create Property-Precedence Graphs. Based on the induced precedence of slot collectives and attributes, the slot precedence and attribute precedence graphs can be created. As described in step 2, for two properties P_1 and P_2, regardless of whether they are attributes or slot collectives, three cases are distinguished:

(1) $P_1 \preccurlyeq P_2$
(2) $P_2 \preccurlyeq P_1$
(3) $P_1 \sim P_2$

Each property added to a property-precedence graph is first wrapped in a property group, resulting in two property groups $P_1 \in PG_1$ and $P_2 \in PG_2$. If two properties are identified as concomitant, i.e., $P_1 \sim P_2$, they should form a shared property group $P_1, P_2 \in PG_s$. Since they are detected as concomitant all relations of P_1 and P_2 to any other property are identical for both properties. The existing adjacency list of the property-precedence graph must then be updated; each property group containing either P_1 or P_2 must be expanded to include the missing property.

If either $P_1 \preccurlyeq P_2$ or $P_2 \preccurlyeq P_1$, updating the precedence graph adjacency list is straightforward. The property group that includes the preceding property P_1 is added as a node to the graph, i.e., as a key to the adjacency list. The property group including the succeeding property P_2 is added as an entry to the value list of the key. If either P_1 or P_2 are part of an already existing property group, the precedence is added for the respective property groups they are a part of.

5 Demonstration of Property-Precedence Analysis with FMML^{x} Models

In this section, the current implementation of the property-precedence analysis is demonstrated using two simplified examples. The first example (Sec. 5.1) consists of a single flat class with eight instances. The second example (Sec. 5.2) consists of two classes that are associated with each other, discussing how property-precedence analysis deals with association ends and slot links.

5.1 Example A: Recovery of Attribute Instantiation Hierarchies

The first example investigates whether property-precedence analysis can recover a multi-level hierarchy solely from attribute values. The example intentionally represents a simple case without contradictory precedence relations in order to illustrate the basic behavior of the algorithm. The example models are also available in the publicly accessible source-code files for ModelDeepener©.

Presentation of Example A. Example A consist of a single class called `RentalCar` with eight instances. The license plate number is different for each instance. The attributes `brandName` and `modelName` are modeled as concomitant, i.e., each brand name co-occurs with exactly one model name and vice versa. All car models are either categorized as a luxury car or a compact car. The instance values available for the class are displayed in Table 1. Based on the provided instance values, the following attribute relations are induced:

- $brandName \sim modelName$
- $brandName \prec rentalCarCategory$
- $licensePlateNumber \preccurlyeq brandName$
- $licensePlateNumber \preccurlyeq modelName$
- $licensePlateNumber \prec rentalCarCategory$
- $modelName \prec rentalCarCategory$

name of object	rental car-category	brand name	model name	license plate number
rc1	Luxury	Audi	A3	lfvo
rc2	Luxury	Ford	Mustang	afj2
rc3	Luxury	Audi	A3	asfj21
rc4	Compact	MINI	Cooper	afi82
rc5	Compact	MINI	Cooper	dfsndf
rc6	Compact	MINI	Cooper	djf73
rc7	Compact	Renault	Twingo	kdf83
rc8	Compact	Renault	Twingo	823rfw

Table 1: Instance values for example class representing rental cars

The resulting attribute-precedence graph is displayed in Fig. 6. The resulting graph reflects the possible multi-level hierarchy. The attribute `rentalCarCategory` precedes the attributes `brandName` and `modelName` because its values characterize larger subsets of instances. The attributes `brandName` and `modelName` are identified as concomitant since every observed brand uniquely determines a model and vice versa. Finally, license plate numbers represent the lowest level because each value identifies a single vehicle. Based on this graph, each attribute can be provided with an instantiation level (see Sec. 2.2 on the difference between instantiation levels and feature potency). The attribute `licensePlateNumber` is different for each particular rental car and should thus be instantiated at level 0. The attributes `brandName` and `modelName` would be redundant if specified separately for each particular rental car and should thus be instantiated at level 1. Finally, the attribute `rentalCarCategory` would be redundant even for the different brand–model combinations and should thus be instantiated a level higher at level 2. To realize this instantiation hierarchy of attributes, an L3 class containing these attributes with the newly proposed instantiation levels needs to be added.

Discussion of Example A. This example illustrates a simple case for property-precedence analysis. The instance values contain no contradictions. It exemplifies that the tool is capable of detecting concomitant attributes. It also illustrates how the induction of attribute precedence may be flawed based on the provided instance values. For each of the provided instances, only one car model exists for one car brand. However, each car brand typically offers a variety of car models. This limitation originates from the provided instance data rather than from the analysis itself. A more comprehensive set of example instances may also include contradicting attribute relations if, e.g., one car brand includes multiple car models and multiple car brands offer models with the same name. Then, the tool is not capable of inducing any unambiguous precedence between both properties. Naming the proposed classes in multi-level models, such as a newly proposed L3 class to represent the hierarchy of rental cars, remains a general challenge of model deepening.

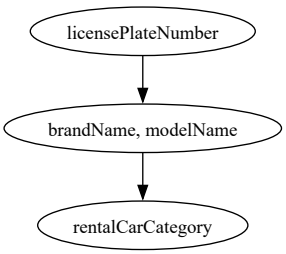

Figure 6: Attribute-precedence graph generated for class `RentalCar` in Example A

5.2 Example B: Extension to Associations

The second example investigates how property-precedence analysis extends from attributes to associations.

Presentation of Example B. Example B also serves to represent rental cars. An excerpt of the model is displayed in Fig. 7. In total, the example contains eight instances of `Car` and four instances of `CarModel`. Each car has a unique license plate number and vehicle identification number (VIN). For the class `CarModel`, the instance values express that the attributes `brandName` and `modelName` are concomitant to each other, as well as precede `rentalCarCategory`. The attribute-precedence graphs for both classes are displayed in Fig. 8. The names `carModel` and `car` in the attribute-precedence graphs correspond to the names of the association ends of the `is_of` association that are not displayed in Fig. 7.

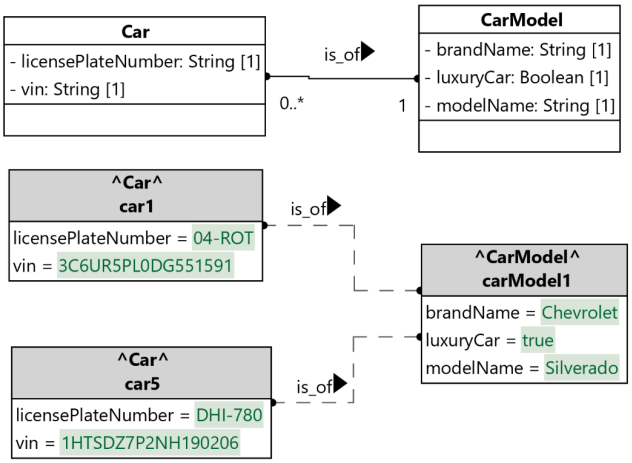

Figure 7: Excerpt of flat input model for example B

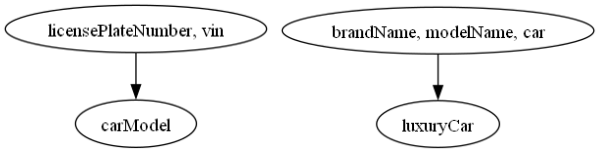

Figure 8: Attribute-precedence graphs generated for Classes `Car` and `CarModel`

Discussion of Example B. Example B illustrates how the current implementation of property-precedence analysis handles associations and links. The two attribute-precedence graphs may be merged by, e.g., swapping the node carModel in the attribute-precedence graph for the class `Car` with the attribute-precedence graph of the class `CarModel`. The multi-level model suggested by such a merged attribute-precedence graph corresponds to the multi-level model suggested by Example A with the addition of the VIN as an attribute that should be instantiated for L0 objects. Example B also illustrates an example of the type-object pattern. The property-precedence analysis as outlined here, however, does not require any type-level heuristics like suffixes of class names or multiplicities of associations, and instead analyzes the provided instance values.

6 Discussion and Remaining Challenges

The proposed property-precedence analysis demonstrates that instance-level information constitutes a valuable source of evidence for identifying possible multi-level hierarchies in flat models. The analysis is more independent of individual modeling styles compared to type-level heuristics and instead focuses on the semantic information implicitly contained in the instance data. Nevertheless, several challenges remain for property-precedence analysis.

Challenge 1: Dependence on available and quality data. The quality of induced multi-level hierarchies depends chiefly on the availability, representativeness, and correctness of instance data. Since precedence relations are inferred from observed property distributions, incomplete or biased instance populations may result in incorrect or missing precedence relations.

Challenge 2: Fragility of proposed multi-level hierarchy. Any proposed multi-level hierarchy depends upon the data available for the analysis at the given time. The abstractions proposed in such a multi-level hierarchy should be time-invariant, i.e., they should represent stable abstractions that remain valid and useful even after the analysis was conducted. The basis of the analysis is, however, restricted to data collected in the past and may not account for probable or expected changes in the future. This is a general challenge of inductive inferences.

Challenge 3: Lack of validation/evaluation of proposed multi-level hierarchies. The analysis approach supports identifying subsets of property characterizations given an instance population. On the basis of the set comparisons of slot values, a candidate multi-level hierarchy is proposed. With regards to challenges 1 and 2, it is important to note that inferred precedence relations should be regarded as hypotheses that require further validation. The analysis approach by itself does not provide any additional evaluation of the proposed multi-level hierarchy, taking into account, for example, guidelines for MLM as presented in [9]. Boolean attributes make this challenge particularly clear. Since they always partition instances into two groups, they therefore tend to induce precedence relations that do not transform into useful multi-level models. Validation may be partly outsourced to domain experts but, given the likely lack of familiarity with MLM, they should also be provided with criteria/guiding questions that help them decide when a new classification level is useful/reasonable.

Challenge 4: Dealing with numerical data. The examples discussed in Sec. 5 only contain categorical data. Categorical data is appealing since it implies the existence of distinct categories that can be identified, compared, and analyzed. Flat models may, however, also contain numerical values. In many cases, it does not make sense to treat numerical values as categories (a counterexample is provided in the example in Sec. 3). For example, imagine that the rental car example discussed in Sec. 5 is expanded to include attributes such as horse power, acceleration speed, and maximum speed. A possible multi-level hierarchy may distinguish between fast powerful cars and regular cars. The identification of such distinct categories by property-precedence analysis demands that adequate intervals of numerical data are formed. Since the particular values are probably different for each particular car, the attributes would still need to be instantiated on level 0. But the identified intervals may be enforced by adding constraints through the respective types (e.g., for validating the minimum acceleration speed of instances).

Challenge 5: Naming of proposed classes in a multi-level hierarchy. Still unexplored remain approaches for the automated naming of newly proposed classes. Even when applied to a single class, property-precedence analysis may suggest multiple new classes and each class must be provided with a name. Here, three subchallenges may be distinguished.

Subchallenge 5.1: Naming the highest-level class. The name of the original flat class may be used for the new highest-level root class. Just like the examples in Sec. 3 and 5.1 the original class name (`HotelRoom` and `RentalCar`) may be conceptually overloaded – the flat class name accounts for particular instances as well as types. When creating a multi-level hierarchy, the name of the original flat class may be adapted for the newly proposed root class. However, just adding the suffix 'Type' would not completely resolve this issue since the instances of the root class may also serve as meta types, as is the case for the rental-car example. Using suffixes like 'MetaMetaType,' depending on the level of the class, would lead to artificial class names.

Subchallenge 5.2: Using slot values for class names. For proposed intermediate classes, the slot values of a class may be used as a new class name. Considering the example from Sec. 5.1, this may lead to class names such as `RenaultTwingo`. While appealing at first, this approach faces two issues. First, if multiple slot values are defined in a type, a simple combination of all values without any reflection of their order and importance may yield to irritating class names. Second, for some slots (like those of Boolean attributes) it may be reasonable to use (parts of) attribute names instead, like in the examples in Sec. 5 for luxury cars.

Subchallenge 5.3: Identifying names of detected concepts Some distinctions identified by property-precedence analysis may correspond to already existing names, like in Sec. 3 'suites.' These names may be absent from the model altogether and must somehow be identified only after the analysis was conducted. This requires the inclusion of external semantic resources.

Some challenges, like challenge 1, are avoided when focusing on type-level heuristics. Challenges 4 and 5 may be counteracted by expanding the current property-precedence analysis and the subsequent deepening transformation with further heuristics and statistical analysis, e.g., to identify statistical distributions of numerical values in the provided instance data and form intervals. External semantic resources, such as structured vocabularies like WordNet or large language models (LLMs) may be consulted, for example, to mitigate challenge 5. LLMs may support property-precedence analysis by generating synthetic instance data and providing edge cases that challenge the proposed multi-level model. The use of LLMs for model deepening is investigated in [23].

7 Conclusion

Model deepening has the potential to facilitate the adoption of multi-level modeling by supporting the automated transformation of flat models into multi-level models. While previous work has primarily focused on transformation mechanisms or the heuristic identification of deepening opportunities, comparatively little attention has been devoted to the underlying analysis problem. This paper proposed *property-precedence analysis*, an instance-driven approach for identifying possible multi-level hierarchies based on available model instance data. Building upon Bunge's notion of property precedence, the proposed analysis reconstructs precedence relations between properties and uses them to derive candidate multi-level hierarchies. The approach was implemented in the prototype ModelDeepener© and demonstrated using two small example models. The presented work presents an initial step towards more comprehensive model-deepening analysis approaches. Several challenges remain, such as handling low-quality data and evaluating the quality of proposed multi-level models. The proposed analysis approach provides a complementary source of evidence for model deepening that may be combined with type-level heuristics to encounter remaining challenges.

Future work should focus on mitigating the identified challenges and integrate property-precedence analysis with existing type-level heuristics into a comprehensive framework for model deepening. It remains an open research question how LLMs may support property-precedence analysis. The proposed analysis approach should also be applied to more comprehensive, real-world use cases in order to uncover further challenges. An evaluation framework with corresponding evaluation criteria, such as potentially algorithmic complexity, should be developed in order to better compare analysis approaches for model deepening.

References

[1] Colin Atkinson, Ralph Gerbig, and Thomas Kühne. 2015. A Unifying Approach to Connections for Multi-Level Modeling. In *MULTI 2015: Proceedings of the 2nd International Workshop on Multi-Level Modeling*. 216–225. doi:10.5555/3351736.3351764

[2] Colin Atkinson and Thomas Kühne. 2001. The Essence of Multilevel Metamodeling. In *UML 2001 - The Unified Modeling Language: 4th International Conference Proceedings*. 19–33. doi:10.1007/3-540-45441-1_3

[3] Mira Balaban, Igal Khitron, and Azzam Maraee. 2018. Context-aware Factors in Rearchitecting Two-Level Models into Multilevel Models. In *Proceedings of MODELS 2018 Workshops*. 683–692.

[4] Mario Bunge. 1977. *Treatise on Basic Philosophy, Volume 3. Ontology I: The Furniture of the World.* D. Reidel Publishing Company, Dordrecht and Boston.

[5] Tony Clark, Paul Sammut, and James Willans. 2015. *Superlanguages: Developing Languages and Applications with XMF.* doi:10.48550/arXiv.1506.03363

[6] Juan de Lara and Esther Guerra. 2018. Refactoring Multi-Level Models. *ACM Transactions on Software Engineering and Methodology* 27, 4 (2018), 1–56. doi:10.1145/3280985

[7] Juan de Lara, Esther Guerra, and Jesús Sánchez Cuadrado. 2014. When and How to Use Multilevel Modelling. *ACM Transactions on Software Engineering and Methodology* 24, 2 (2014).

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

[9] Ulrich Frank. 2021. Prolegomena of a Multi-Level Modeling Method Illustrated with the FMMLx. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 521–530. doi:10.1109/MODELS-C53483.2021.00081

[10] Ulrich Frank. 2022. Multi-Level Modeling: Cornerstones of a Rationale. *Software and Systems Modeling* 21 (2022), 451–480. doi:10.1007/s10270-021-00955-1

[11] Ulrich Frank and Daniel Töpel. 2020. Contingent Level Classes: Motivation, Conceptualization, Modeling Guidelines, and Implications for Model Management. In *MODELS Companion '20*. doi:10.1145/3417990.3421413

[12] Ulrich Frank and Daniel Töpel. 2024. Association Types: Motivation, Specification and Implementation with the XModelerML. In *MODELS Companion '24*. 780–789. doi:10.1145/3652620.3688210

[13] Thomas R. Gruber. 1995. Toward Principles for the Design of Ontologies Used For Knowledge Sharing? *International Journal of Human-Computer Studies* 43, 5-6 (1995), 907–928. doi:10.1006/ijhc.1995.1081

[14] Gustavo L. Guidoni, Joao Paulo A. Almeida, and Giancarlo Guizzardi. 2022. Preserving Conceptual Model Semantics in the Foward Engineering of Relational Schemas. *Frontiers in Computer Science* 4 (2022). doi:10.3389/fcomp.2022.1020168

[15] Manfred A. Jeusfeld and Ulrich Frank. 2021. Unifying Multi-Level Modeling: A Position Paper. In *MODELS Companion '21*. 536–540. doi:10.1109/MODELS-C53483.2021.00083

[16] Thomas Kühne. 2018. Exploring Potency. In *18th Intenational ACM/IEEE Conference on Model Driven Engineering Languages and Systems*. 2–12. doi:10.1145/3239372.3239411

[17] Thomas Kühne, Joao Paulo A. Almeida, Colin Atkinson, Manfred A. Jeusfeld, and Gergely Mezei. 2023. Field Types for Deep Characterization in Multi-Level Modeling. In *MODELS Companion '23*. 639–648. doi:10.1109/MODELS-C59198.2023.00105

[18] Thomas Kühne and Pierre Maier. 2024. FMMLx and DLM: A Contribution to the MULTI Collaborative Comparison Challenge. In *MODELS Companion '24*. 800–809. doi:10.1145/3652620.3688212

[19] Amruth N. Kumar, Raj, Rajendra, K., Herif G. Aly, Monica D. Anderson, Brett A. Becker, Richard L. Blumenthal, Eric Eaton, Susan L. Epstein, Michael Goldweber, Pankaj Jalote, Douglas Lea, Michael Oudshoorn, Marcelo Pias, Susan Reiser, Christian Servin, Rahul Simha, Titus Winters, and Qiao Xiang. 2023. Computer Science Curricula. doi:10.1145/3664191

[20] Arne Lange, Ulrich Frank, Colin Atkinson, and Daniel Töpel. 2023. Comparing LML and FMMLx: A Contribution to the MULTI Collaborative Comparison Challenge. In *MODELS Companion '23*. 669–678. doi:10.1109/MODELS-C59198.2023.00108

[21] Fernando Macías, Esther Guerra, and Juan de Lara. 2017. Towards Rearchitecting Meta-Models into Multi-Level Models. In *36th International Conference, ER 2017 Proceedings*. 59–68. doi:10.1007/978-3-319-69904-2_5

[22] Fernando Macías, Adrian Rutle, and Volker Stolz. 2018. A Tool for the Convergence of Multilevel Modelling Approaches. In *Proceedings of MODELS 2018 Workshops*, Regina Hebig and Thorsten Berger (Eds.). 633–642.

[23] Pierre Maier and Vicky Kadziolka. 2026. Model Deepening with Large Language Models: Insights from Exploratory Studies with ChatGPT. In *ER 2025 Workshops Proceedings*. 119–136. doi:10.1007/978-3-032-08620-4_8

[24] Pierre Maier and Tobias Schwarz. 2024. UML++: Enhancing Student Learning of Object-Oriented Modeling through Executable Objects. In *MODELS Companion '24*. 107–114. doi:10.1145/3652620.3687777

[25] Pierre Maier and Daniel Töpel. 2025. XModelerML v3: Integrating Executable UML with a Multi-Level Language Engineering, Modeling, and Execution Environment. In *MODELS Companion '25*. doi:10.1109/MODELS-C68889.2025.00024

[26] Bernd Neumayr and Michael Schrefl. 2022. Domain Object Hierarchies Inducing Multi-Level Models. *Software and Systems Modeling* 21 (2022), 587–621. doi:10.1007/s10270-022-00973-7

[27] Jeffrey Parsons and Yair Wand. 2003. Property-Based Semantic Reconciliation of Heterogeneous Information Sources. In *Conceptual Modeling – ER 2002. 21st International Conference on Conceptual Modeling Proceedings*. Springer, Berlin and Heidelberg, 351–364. doi:10.1007/3-540-45816-6_34

[28] Daniel Töpel and Björn Benner. 2017. Maintenance of Multi-Level Models: An Analysis of Elementary Change Operations. In *Proceedings of MODELS 2017 Satellite Events*. 243–250.

[29] Daniel Töpel and Monika Kaczmarek-Heß. 2022. Towards Flexible Creation of Multi-Level Models: Bottom-Up Change Support in the Modeling and Programming Environment XModeler. In *MODELS Companion '22*. 404–413. doi:10.1145/3550356.3561553