2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)

2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) | 979-8-3315-7990-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/MODELS-C68889.2025.00024

XModeler^{ML} v3: Integrating Executable UML with a Multi-Level Language Engineering, Modeling, and Execution Environment

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

Daniel Töpel
University of Duisburg-Essen
Essen, Germany
daniel.toepel@uni-due.de

***Abstract*—We present XModeler^{ML} v3, a multi-level modeling tool that is based on the Flexible Multi-Level Modeling and Execution Language (FMML^{x}). Multi-level modeling enables the representation of domain knowledge at higher levels of classification, thus reducing conceptual redundancy and improving the integrity, adaptability, and comprehensibility of domain models. This furthermore allows for the specification of domain-specific languages using more generic domain-specific languages, rather than starting from scratch every time. The XModeler^{ML} provides a shared representation of model and code and can be used to instantiate and execute models at runtime. With version 3, we now offer the opportunity to specify executable UML diagrams in XModeler^{ML} using a specifically developed dialect of FMML^{x} called UML++. In this demonstration, we showcase how XModeler^{ML} v3 integrates multi-level and two-level modeling modes. We describe practical modeling scenarios where this integrated approach offers clear advantages and demonstrate how users can define, execute, and evolve executable models seamlessly within the tool.**

***Index Terms*—multi-level modeling, XModeler^{ML}, FMML^{x}, bottom-up modeling, language engineering**

I. Multi-Level Modeling: Prospects and Challenges

For over twenty years, multi-level modeling languages have been proposed as an advancement to the traditional two-level modeling languages commonly used, such as the UML or the ER modeling language [1]. Unlike conventional, two-level modeling, multi-level modeling (MLM) provides the following features [2]: First, it allows for an unbounded number of classification levels. Second, the traditional class-object dichotomy is alleviated, allowing modeling constructs to possess both class and object facets simultaneously. Every class is an object, which enables classes to have states and be executed. Third, the instantiation of class properties is not restricted to the level immediately below but may also be deferred to lower levels. This allows, for example, to model a class `Product` at level 2 with an attribute `id: String` that is instantiated two levels beneath at level 0, thus avoiding the need to specify the attribute redundantly for all instances of `Product` at level 1. Multiple publications discuss in detail how these language features may aid in improving the reusability, adaptability, and comprehensibility of conceptual models, among other prospects (cf. [3], [4]).

Despite these undisputed benefits, MLM has yet to establish itself as a mainstream approach to conceptual modeling, lacking broad industry adoption and serious standardization efforts. This situation is puzzling, not least because multi-level modeling languages constitute a monotonic extension to the traditional modeling paradigm. Any model that can be represented with a two-level language can also be represented using a multi-level language; however, multi-level languages offer additional features that may allow for a more efficient reconstruction of the domain. Consider, for example, the UML class diagram displayed in Fig. 1 and the same model reconstructed with the multi-level modeling language FMML^{x} in Fig. 2. Each UML class is modeled as an FMML^{x} class on level 1 (L1)[1] as shown by the integer value next to the class name. The instantiation levels of all attributes and association ends, displayed as a white number against a black background, are 0.

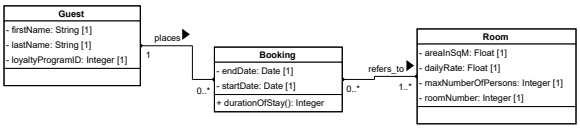

Fig. 1. Example model of hotel room bookings modeled as a UML class diagram.

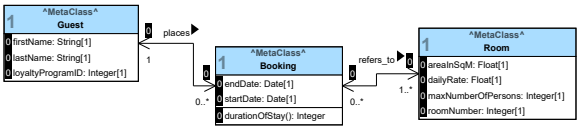

Fig. 2. Example model from Fig. 1 modeled with FMML^{x}.

[1]Note that we use the abbreviation L*n* to refer to levels in a multi-level context as opposed to the common M*n* abbreviation used in the MOF architecture since the semantics of a level are different from that in MOF.

979-8-3315-7990-6/25/$31.00 ©2025 IEEE
DOI 10.1109/MODELS-C68889.2025.00024

Authorized licensed use limited to: University Duisburg Essen. Downloaded on May 12,2026 at 22:13:39 UTC from IEEE Xplore. Restrictions apply.

Using a multi-level modeling language, we can re-engineer this model by promoting the class `Room` to L2 and distinguishing between different types of rooms on L1 (Fig. 3). In MLM terminology, the L1 objects `StandardRoom` and `JuniorSuite` are *concretions* of `Room` [5, p. 2]. As objects, they serve as instances of `Room` and may provide values for attributes. As classes, they serve as specializations of `Room` and inherit its properties. The strict dichotomy between classification and generalization, present in the UML, is alleviated.

We have been actively involved in a long-standing research project aimed at furthering the development of multi-level modeling [6]. As part of this research project, the multi-level modeling language FMML^{x} (Flexible Multi-Level Modeling and Execution Language) and its supporting toolset, XModeler^{ML}, have been developed. Much research from the past years has contributed to MLM by providing even more advanced language features.Yet, we observe that – despite growing technical maturity – MLM remains confined to niche academic and industrial contexts. We assume that this adoption barrier is not caused by conceptual deficiencies of MLM, but rather by its high entry barrier and its steep learning curve.

To counteract this issue, we set out to lower the entry barriers to MLM by supporting a seamless transition between two-level and multi-level modeling modes in XModeler^{ML}. Specifically, we implemented support for two usage scenarios:

1) *From Deep to Flat Modeling.* Expert language engineers can define deep, multi-level meta models, while domain experts only create and manipulate models using a familiar, easy-to-comprehend notation that fades out the complexity of the underlying meta model.
2) *From Flat to Deep Modeling.* Modelers can start with flat, two-level models and incrementally deepen their models by introducing new classification levels as needed, thus "growing into" MLM without committing upfront.

These scenarios are now supported in XModeler^{ML} v3. In this demonstration paper, we present the conceptual underpinnings, key features, and usage scenarios of the new release. We anticipate that these new features offer a path toward greater acceptance and dissemination of multi-level modeling in practice.

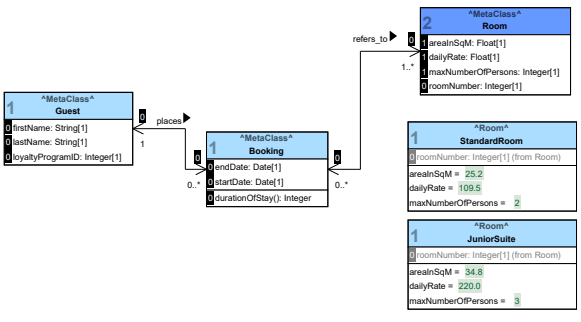

Fig. 3. Example model from Fig. 1 re-engineered as deep FMML^{x} model.

II. Overview of XModeler^{ML} v3

The original XModeler was developed as a language-engineering environment built on top of XCore, the meta-circular language kernel of the executable meta-modeling facility (XMF) [7]. The FMML^{x} is developed as a monotonic extension to XCore, offering specific multi-level modeling features such as deferred instantiation. The XModeler^{ML} is the modeling and execution environment for the FMML^{x}. It is an open-source program that can be downloaded at www.le4mm.org/xmodelerml.

Besides XModeler^{ML}, MetaDepth [8] and Melanee [9] are available multi-level modeling tools that each support a different multi-level modeling language. Each tool varies significantly in terms of the functionalities it offers. MetaDepth, for example, is a command-line tool and does not offer any diagram editor. Unlike other MLM languages and tools, the XModeler^{ML} allows for the instantiation and execution of FMML^{x} models at runtime. Custom constraints and operations in FMML^{x} are defined using the XOCL programming language [10].

Motivated by limitations of the UML, especially with regard to teaching and learning object-oriented languages, we developed a dialect of FMML^{x}, called UML++, which offers an executable version of core UML concepts at runtime [11]. The diagram in Fig. 1 is created with UML++. Users can create UML++ diagrams using the modeling and execution environment UML-MX© [12], a specifically implemented variant of the XModeler^{ML} tailored to educational purposes.

UML++ is different in some regards from FMML^{x}. Specific language features of FMML^{x} required for multi-level modeling, such as the classification level of a class or the instantiation level of class properties, are faded out. The notation is slightly adapted to replicate standard UML notation. A list of minor changes can also be noted between standard UML class and object diagrams and UML++. For example, UML++ includes native support for delegation associations and does not provide notational support for association classes. A detailed list of these differences is discussed in [11]. The diagram editor is also different for UML++ diagrams: all tool features required only for multi-level modeling are invisible.

We assume UML++ is not only well-suited for teaching and learning object-oriented modeling with UML, but may also serve as a point of entry to access features of multi-level modeling languages. Therefore, we set out to integrate the functionalities offered by UML-MX© into the XModeler^{ML}. As a result, UML++ diagrams can now be created and managed using the regular XModeler^{ML}. Users can navigate and manage all models from within the *Control Center* (Fig. 4). As seen in the screenshot, users can choose to create a UML++ or FMML^{x} diagram after having created a model, each using a slightly different diagram editor.

Authorized licensed use limited to: University Duisburg Essen. Downloaded on May 12,2026 at 22:13:39 UTC from IEEE Xplore. Restrictions apply.

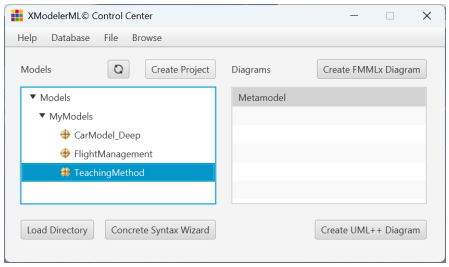

Fig. 4. XModeler^{ML} v3 Control Center.

III. From Deep to Flat Modeling: Multi-Level Domain-Specific Languages for Two-Level Models

A. Motivation and Description of Scenario

The advantages of domain-specific languages over general-purpose languages like UML have been discussed for long (e.g., [13]). The key benefit is that domain-specific languages incorporate domain knowledge directly into the language, thereby promoting modeling productivity as well as model quality. Since general-purpose languages like UML only specify generic elements, such as `Class` or `Attribute`, they are well-suited to cover all sorts of domains, but they also allow modeling any sort of gibberish, too.

Multi-level modeling disrupts the classical dichotomy between language development and application. While a domain-specific language is typically specified through a meta model on one classification level, namely M2, and applied only on the classification level immediately below, namely M1, in multi-level models, the language specification may be scattered across multiple levels from level 1 to any highest level desired.

Given that complex meta models may be too demanding for common domain modelers, it should be possible to preserve the conventional two-level modeling mode for them while allowing expert language engineers to specify advanced multi-level meta models. For this purpose, XModeler^{ML} must enable domain modelers to use domain-specific languages while hiding the complexities of the underlying multi-level meta model.

B. Core Features and Demonstration of Scenario in XModeler^{ML} v3

A language engineer may develop a multi-level meta model to manage rental cars as displayed in Fig. 5. The language specification is scattered across multiple classification levels. In this language specification, language engineers may use any multi-level language features of the FMML^{x}. They can specify classes at levels higher than 1 and 2, such as `RentalCar`. These classes may make use of deferred instantiation, as exemplified by attributes such as `numberOfSeats: Integer`, instantiated for each car model at L1, or `licensePlateNumber: String`, instantiated for each particular car at L0. Classes may also specify values for attributes, as can be seen in the L2 concretions of `RentalCar`. The DSML is not restricted to level 2 and, furthermore, includes operations and constraints that will be executed/validated at runtime.

Regular domain modelers may create flat models based on this language specification. By creating a UML++ diagram for this DSML, higher-level classes and class properties are faded out. Users may choose between standard UML notation or a domain-specific concrete syntax if one is specified. They can access the domain-specific concepts specified in the meta model. If any syntactical rules are violated, the tool notifies the user. For example, if the end of the rental period is set earlier than the start, the constraint `rentalPeriodCorrect` is violated and the user is notified. Similarly, users are notified if the multiplicity of an association or attribute is violated. The model created by a standard domain modeler is also executed at run time. It therefore provides dynamic return values for all specified operations. The operation `isCurrentlyRented(): Boolean` in `RentalCar`, for example, shows whether a specific car is currently part of an active rental agreement. The object diagram in Fig. 6 illustrates an example instantiation of the meta model in Fig. 5. The diagram was created on Jul 10, 2025, which is why the rental status is computed as "Coming". In the example model, the constraint `paymentIsOverdue` is violated. Therefore, a custom error message runs beneath the name compartment of the object `invoice1`.

IV. From Flat to Deep Modeling: Bottom-Up Modeling and Class Migration

A. Motivation and Description of Scenario

The construction of conceptual models typically follows a top-down approach: classes are defined first, and objects may be instantiated next. This same principle holds in the design

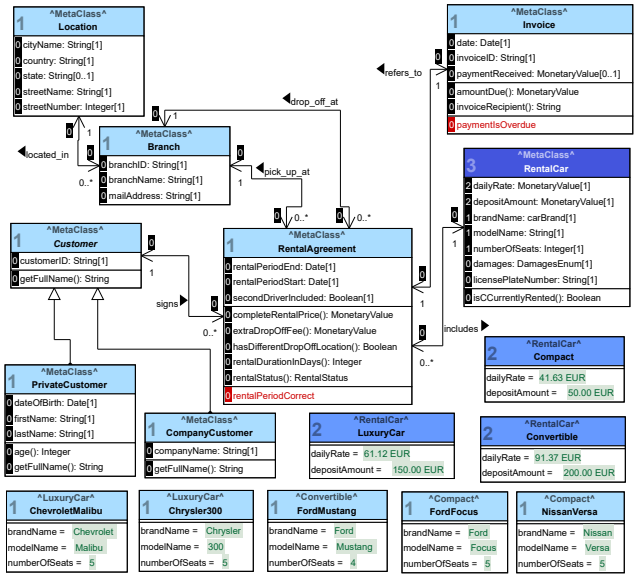

Fig. 5. Multi-level meta model to manage rental cars created with XModeler^{ML} v3.

Authorized licensed use limited to: University Duisburg Essen. Downloaded on May 12,2026 at 22:13:39 UTC from IEEE Xplore. Restrictions apply.

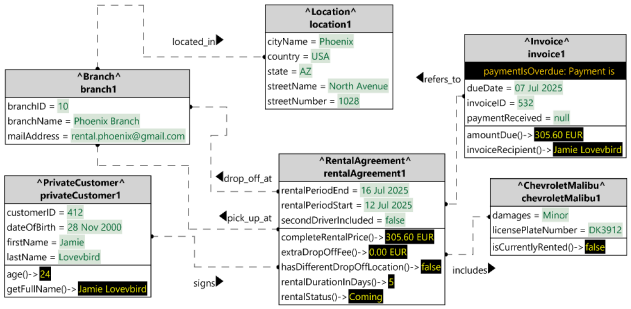

Fig. 6. Object diagram for multi-level model from Fig. 5

and use of domain-specific languages. In this context, language engineers first specify the language, usually by defining a meta model that introduces the domain-specific concepts, after which domain modelers use the language by instantiating the provided language constructs. This top-down approach is also followed in Sec. III.

The top-down modeling approach is enforced by class-based object-oriented languages, which require the specification of a class before instantiating any of its objects. As a result, modelers are forced to commit early to class definitions, even when the domain knowledge is still evolving or incomplete. This impedes exploratory modeling and the incremental construction of models. This problem is aggravated in multi-level modeling, where it is necessary to model the highest-level classes first before instantiating more specific types. To counteract this issue, modelers should be enabled to add lower-level objects before adding respective domain-specific classes. This modeling approach has been termed *bottom-up modeling* in [14].

We see two principal approaches to support bottom-up modeling: class promotion and class migration. *Class promotion* entails "lifting up" a class to a higher level and potentially adjusting the instantiation levels of its properties in the process. The re-engineering example in Sec. I demonstrates this approach. *Class migration* entails changing the class of an object, allowing objects to migrate to higher-level domain-specific classes. In this demonstration paper, we focus on bottom-up modeling via class migration. The implementation of class promotion is subject to current work.

B. Core Features and Demonstration of Scenario in XModeler^{ML} v3

In FMML^{x}, all user-specified elements are objects and ultimately instances of `MetaClass`. Domain-specific classes added by a user are direct instances of `MetaClass`. If a user wants to initially start by constructing objects and classes at levels L0 and L1 alone (i.e., by constructing a regular UML++ diagram), and only afterwards add higher-level domain-specific classes that the original L1 classes shall be instances of, class migration must be performed. Bottom-up modeling can then be realized by performing three successive steps after an initial two-level model has been created:

1) Create a new domain-specific class at L2 that serves as the new class for some of the current L1 objects. Note that L1 objects serve as regular classes in UML++ diagrams.
2) Change the class of each L1 object. This can be done by right-clicking on the object and selecting the respective menu item. This migrates the object from its original class (`MetaClass`) to the newly added, domain-specific class.
3) Analyze and refactor the properties of the instances of the new class.

This three-step process is also illustrated in Fig. 7. To refactor properties of class instances, a separate dialog is provided to users (see Fig. 7B). The dialog lists all attributes of the instance classes. For each attribute users must choose between the following options:

- *Keep.* Keeps the attribute in the current L1 object without any modifications.
- *Keep as.* Keeps the attribute in the current L1 object. The attribute may be renamed to avoid potential naming conflicts with new attributes of the L2 class.
- *Drop.* Removes the attribute from the model.
- *Pull up.* Lifts the attribute to the new L2 class. The original instantiation level of the attribute, which is zero for an L1 class, is maintained. All pre-existing references to the attributes are updated. Through pulling up attributes to the new L2 class, they are inherited in all concretions of this class.
- *Merge into.* Attribute from L1 class shall be merged into an attribute of the L2 class. In this case, the corresponding L2 attribute that the L1 attribute should be merged into must be selected from a drop-down menu. This is visible in Fig. 7B in the "Merge With" column.

A row is highlighted in red if any conflicts are detected, for example, if two attributes with the same name are to be promoted to the L2 class.

Although this process yields a new classification level, note that, at first, only inheritance between L2 and L1 classes is present (see the resulting model in Fig. 7C), which could also be realized by adding a generalization relationship. But through the migration of `TomCat` and `IIS` to the new domain-specific class `SoftwareServer` modelers can make use of instantiation semantics, too, for example by adding new attributes to `SoftwareServer` that are instantiated for each L1 software server type.

V. Conclusion and Future Research

In this demonstration paper we presented central features of XModeler^{ML} v3, a multi-level modeling and execution environment that now includes support for a seamless transition between two-level and multi-level modeling modes. With this release, XModeler^{ML} enables both expert language engineers and domain modelers to work within a shared environment while engaging at different levels of abstraction.

Two usage scenarios serve to illustrate the practical value of these capabilities. In the first scenario, we have outlined how,

Authorized licensed use limited to: University Duisburg Essen. Downloaded on May 12,2026 at 22:13:39 UTC from IEEE Xplore. Restrictions apply.

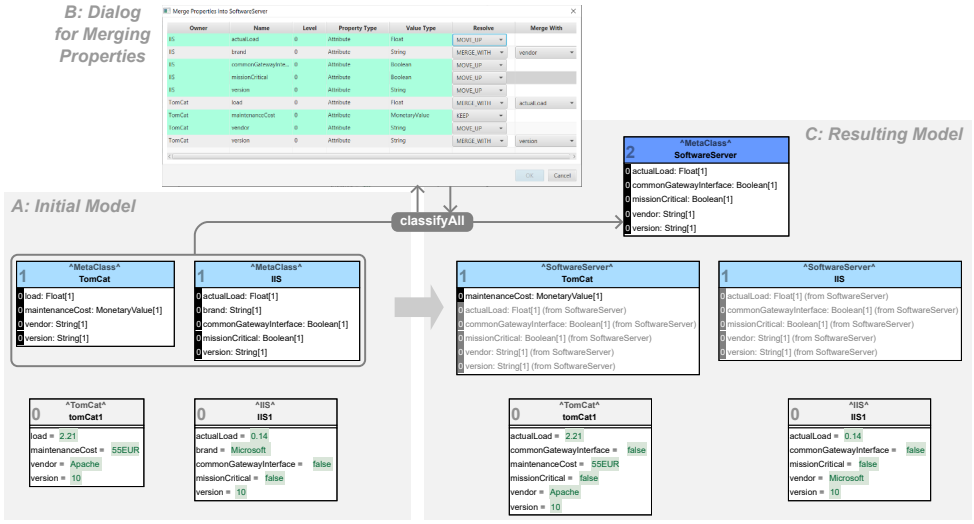

Fig. 7. Illustration of bottom-up modeling approach with class migration implemented in XModeler^{ML} v3 [14]

through the integration of FMML^{x} and UML++, expert users can define deep multi-level meta-models while a simplified, flat representation can be offered to common domain users. In the second scenario, we have shown how users can incrementally evolve flat models by introducing new classification levels through class migration. We believe that both scenarios are well-suited to lower the entry barriers to multi-level modeling and may therefore help in realizing often proclaimed benefits of multi-level modeling.

Future work will focus on expanding the modeling capabilities introduced in this version. One key direction involves the implementation of class promotion, which allows users not only to migrate instances but also to lift up classes to higher classification levels. Additionally, we are investigating how different bottom-up modeling process steps may be automated, e.g., by automatically detecting and resolving conflicts while merging class properties [15]. The capabilities offered by this latest version may also aid teaching and learning model-driven software engineering and multi-level modeling [16]. From a usability standpoint, future work should also address how other flat representations (such as UML models created with different tools) may be imported and transformed into UML++ diagrams.

REFERENCES

[1] U. Frank, "Multilevel modeling: Toward a new paradigm of conceptual modeling and information systems design," *Business and Information Systems Engineering*, vol. 6, no. 6, pp. 319–337, 2014.

[2] M. A. Jeusfeld and U. Frank, "Unifying multi-level modeling: A position paper," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pp. 536–540, 2021.

[3] C. Atkinson and T. Kühne, "Reducing accidental complexity in domain models," *Software and Systems Modeling*, vol. 7, pp. 345–359, 2008.

[4] U. Frank, "Multi-level modeling: Cornerstones of a rationale," *Software and Systems Modeling*, vol. 21, pp. 451–480, 2022.

[5] U. Frank and D. Töpel, "Contingent level classes: Motivation, conceptualization, modeling guidelines, and implications for model management," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020.

[6] U. Frank and T. Clark, "Language engineering for multi-level modeling (le4mm): A long-term project to promote the integrated development of languages, models and code," in *Proceedings of the Research Projects Exhibition Papers Presented at the 35th CAiSE*, pp. 97–104, 2023.

[7] T. Clark and J. Williams, "Software language engineering with xmf and xmodeler," in *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments* (M. Mernik, ed.), pp. 311–340, Hershey, PA: IGI Global, 2013.

[8] J. de Lara and E. Guerra, "Deep meta-modelling with metadepth," in *Objects, Models, Components, Patterns: TOOLS 2010 Proceedings* (J. Vitek, ed.), pp. 1–20, Berlin and Heidelberg: Springer, 2010.

[9] C. Atkinson and R. Gerbig, "Melanie: Multi-level modeling and ontology engineering environment," in *Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards*, 2012.

[10] T. Clark, P. Sammut, and J. Willans, *Superlanguages: Developing Languages and Applications with XMF*. Sheffield: Ceteva, 2008.

[11] P. Maier and T. Schwarz, "Uml++: Enhancing student learning of object-oriented modeling through executable objects," in *MODELS Companion '24: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, pp. 107–114, 2024.

[12] U. Frank and P. Maier, "Uml-mx: Boosting power of object-oriented modeling and enriching user experience," in *Enterprise Design, Operations, and Computing. EDOC 2024 Workshops*, pp. 276–286, Springer, 2025.

[13] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Hoboken, NJ: John Wiley & Sons, 2008.

[14] D. Töpel and M. Kaczmarek-Heß, "Towards flexible creation of multi-level models: Bottom-up change support in the modeling and programming environment xmodeler," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pp. 404–413, 2022.

[15] P. Maier and V. Kadziolka, "Model deepening with large language models: Insights from exploratory studies with chatgpt," in *Advances in Conceptual Modeling: - ER 2025 Workshops, CMLS, FCM, LLM4Modeling, OntoCom, and QUAMES, Poitiers, France, October 20–23, 2025, Proceedings*, Springer, 2025.

[16] U. Frank and P. Maier, "From the lowlands of uml to the highlands of mlm: An evolutionary, tool-supported multi-level method for teaching multi-level modeling," in *ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2025.

Authorized licensed use limited to: University Duisburg Essen. Downloaded on May 12,2026 at 22:13:39 UTC from IEEE Xplore. Restrictions apply.