Software and Systems Modeling
https://doi.org/10.1007/s10270-021-00955-1

EXPERT VOICE

Multi-level modeling: cornerstones of a rationale

Comparative evaluation, integration with programming languages, and dissemination strategies

Ulrich Frank[1]

Received: 24 May 2020 / Revised: 17 October 2021 / Accepted: 19 November 2021

Abstract
This expert voice paper presents a comprehensive rationale of multi-level modeling. It aims not only at a systematic assessment of its prospects, but also at encouraging applications of multi-level modeling in business information systems and at providing a motivation for future research. The assessment is developed from a comparison of multi-level modeling with object-oriented, general-purpose modeling languages (GPMLs) and domain-specific modeling languages (DSMLs). To foster a differentiated evaluation, we propose a multi-perspective framework that accounts, among others, for essential design conflicts, different types of users, as well as economic aspects. Besides the assessment of the additional abstraction offered by multi-level modeling, the evaluation also identifies specific drawbacks and remaining challenges. Based on the results of the comparative assessment, in order to foster the adoption and further development of multi-level modeling, we discuss the prospects of supplementing multi-level modeling languages with multi-level programming languages and suggest possible dissemination strategies customized for different groups of users. The paper concludes with an outline of future research.

Keywords Essential design conflicts · Multi-perspective evaluation framework · Multi-level programming languages · Integration of models and code · Multi-level dissemination strategies

1 Introduction

Introduced about 20 years ago [8], with ancestors that go back even further, cf. [56,63,91,95], multi-level modeling has not yet made it to the research mainstream, neither in software engineering nor in conceptual modeling. Nevertheless, it has been persistent, and there are signs of an increased awareness of multi-level modeling, albeit to a modest degree only. For instance, in recent years, the MULTI workshop has developed into an established workshop series within the *Models* conference.[1] A theme issue [10] and a special issue [102] underline the attractiveness of the topic. Also, a recent Dagstuhl seminar [2] confirms the relevance and topicality of multi-level modeling. Last but not least, the small international community of dedicated researchers is still excited about the prospects of multi-level modeling.

From an external perspective, the research field is likely to give a confusing picture: a seemingly marginal topic, but persistent; enthusiastic researchers, but hardly any dispute with followers of traditional conceptual modeling approaches. These traditional approaches are limited to models, the classes of which are all on the same classification level. Therefore, one could refer to them as "single-level" approaches. At the same time, corresponding language architectures, such as, e.g., the metaobject facility (MOF) [92], are also often used for the creation of models on M1 and on M0, the former being usually the main focus. That would justify to speak of "two-level" approaches, which is often the case in publications on multi-level modeling, cf. e.g., [9]. In order to avoid confusion, we shall use the term "traditional approach" whenever we refer to modeling within a language architec-

[1] http://www.modelsconference.org.

Communicated by Adrian Rutle and Manuel Wimmer.

✉ Ulrich Frank
ulrich.frank@uni-due.de

[1] Information Systems and Enterprise Modelling Research Group Institute for Computer Science and Business Information Systems (ICB), University of Duisburg-Essen, Essen, Germany

Published online: 04 January 2022

ture that allows only for models on one specific classification level.

For those who work in the field of multi-level modeling, but even more so for others who wonder whether they should join in, the question emerges as to *whether multi-level modeling actually offers convincing benefits that make a clear difference relative to traditional approaches*. Is it comparable to one of those exotic programming languages that are worshiped by a small group of devotees, who are not interested in spending much time with evangelizing users of other languages? Or is it rather a sleeping giant, the power of which has been underrated for a long time? It is remarkable, though, that there has hardly been any dispute with followers of traditional approaches to conceptual modeling. Instead, the multi-level modeling community lives in a state of peaceful co-existence with followers of traditional approaches. While this situation may be seen as comforting by some, we regard it as ambivalent. On the one hand, an intensive dispute with researchers of traditional approaches to conceptual modeling may be seen as stressful and daunting by some, and might have contributed to an erosion of the community. On the other hand, a field of research can only flourish if it is in competition with alternative approaches. This situation constitutes an opportunity and also, as we believe, the need for action. As far as we can see, the users and followers of traditional approaches do not form a hostile environment. Most members of the multi-level modeling community are respected members of other communities, too. These are favorable conditions for convincing others of the prospects of multi-level modeling. At the same time, it is essential for the future of multi-level modeling to develop a comprehensible rationale in order to promote not only interest in multi-level modeling, but also a much needed debate on its potential and specific challenges.

Within the multi-level modeling community, there seems to be a wide consensus about the pivotal advantage offered by multi-level models, that is, the reduction of "accidental complexity," cf. [9]. Accidental complexity emerges, if the analysis of a domain leads to abstractions that cannot be expressed by the modeling language in use. In other words, we have relevant knowledge about a domain we want to model, but the modeling language does not allow for expressing it. The following example illustrates this problem. It seems to be straightforward to assume that a *printer model* is a *printer*, which is a *peripheral device*. Furthermore, a particular printer is of a certain printer model. Figure 1 shows that representing this fairly simple conceptualization with a traditional object model is problematic—mainly because the predicate *is a* is overloaded. In part, it represents specialization, e.g., with respect to the attribute **`resolution`** in **`Printer`** that is added to the list of inherited attributes. At the same time, **`output`** is instantiated, which indicates an instantiation relationship.

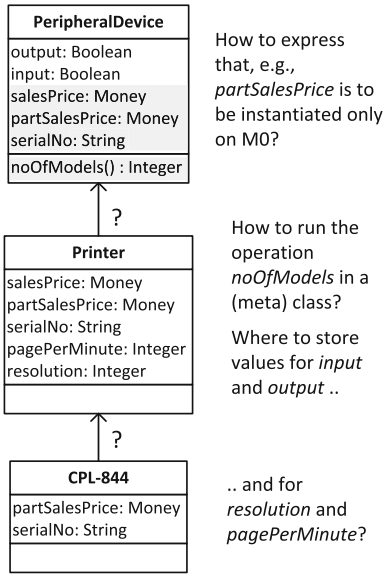

Fig. 1 Illustration of limitations of traditional object-oriented languages

The lack of abstraction traditional approaches like the UML suffer from, leads to the need to overload one classification level with multiple levels of abstraction, which leads to conceptual redundancy and, hence, a threat to integrity. In addition, such workarounds are likely to "obscure the meaning of a domain model […]," which makes it difficult to distinguish between "*accurate reflections* of the *problem domain* […] and just accidental properties of the particular *workaround*" [9, p. 346]. The additional abstraction enabled by multi-level modeling is suited to relax this kind of avoidable model complexity in many cases (see the corresponding examples in Fig. 2).

A considerable number of publications demonstrated clear advantages of applying multi-level modeling to particular use cases, e.g., [18,27,31,32,44,60,65,67,90,98,100], and supported Atkinson's and Kühne's claim that "there is a strong demand for some kind of technology supporting ontological multi-level modeling" [9, p. 357]. However, while various approaches to multi-level modeling are available, it seems still to be the case what the authors stated 12 years ago: "[…] the software modeling community has not taken advantage of them yet" [9, p. 357].

In order to contribute to changing this state of affairs, this paper aims at presenting a comprehensive rationale of prospects and challenges of multi-level modeling, that is, providing a convincing justification of its advantages over other modeling approaches, as well as identifying open issues that still need to be addressed. We do not intend to show that multi-level modeling is a silver bullet. Instead, the intended rationale must also take into account the conditions under which multi-level modeling is advantageous. While our main focus is on modeling languages, we also account for program-

ming languages. Therefore, we also use the more general terms "multi-level language" or "multi-level language architecture" where they seem more appropriate. Our research reported on in this paper is therefore driven by the following questions:

- What perspectives should be accounted for when comparatively evaluating various modeling approaches?
- What are the specific benefits and specific challenges of multi-level modeling compared to other approaches?
- What is needed in order to support the further development and adoption of multi-level modeling?

To reach our objectives, firstly, we present a *multi-perspective* assessment framework, which is subsequently used to compare multi-level modeling to traditional modeling approaches. By accounting for multiple perspectives in our assessment, we move beyond typical engineering-oriented justifications, focusing equally on the perspective of (potential) users (e.g., in terms of a cognitive fit with natural language use), principal system design conflicts, and an economic perspective. With our multi-perspective assessment, we thus point out the various prospects brought about by multi-level modeling. Nevertheless, multi-level modeling also comes with various challenges that include the need for attractive use cases and reaching a critical mass of researchers and users. Against this background, we discuss the benefits of a common representation of multi-level modeling languages and multi-level programming languages at run-time and outline prototypical dissemination strategies for different user groups.

Developing a convincing justification for languages or linguistic artifacts is confronted with serious methodological challenges. In recent years, empirical studies, which are of pivotal relevance in the natural sciences, have become increasingly important in software engineering and business information systems alike. Empirical studies can be a powerful instrument of scientific inquiry. That, however, requires a theory that is suited to explain the existence or variation of certain phenomena. In other words, it should allow for prediction. To serve that purpose, relevant properties of the observed phenomena have to be operationalized in order to allow for measurement. Hypotheses that are deduced from a theory are then tested against the data that resulted from the measurement. In the strict sense of falsification [96], a theory has to be refuted, if a hypotheses derived from it failed. Usually, the verdict is not that rigorous, especially in psychology and the social sciences. Nevertheless, empirical tests do not make sense without the existence of a theory. Take, for example, a construct like "ease of use" that is operationalized by a few hypotheses, which are then tested by having two groups of students work with two different tools. Without a theory, the result may at best be useful for tool vendors to improve their products, but it could hardly be generalized and applied to other tools. To the best of our knowledge, there is no theory in the context of applying conceptual modeling that is powerful enough to explain or even predict the outcome of using a certain modeling approach. Therefore, empirical studies that focus on behavioral aspects of developing and using conceptual models are not an option for our purpose. Note that this is not at all meant to exclude empirical aspects of modeling from our investigation. On the contrary, it is definitely not sufficient to evaluate an approach to *conceptual* modeling through a formal analysis only.

Against this background, we decided for the following configuration of our research approach. To enable a thorough comparison, we introduce a multi-perspective evaluation framework. It accounts, among others, for abstraction concepts, essential design conflicts, the user perspective, and economic aspects. To account for design and engineering views we justify our assessment with presuppositions that are likely consensual, meaning presuppositions that are largely agreed upon in discourse among a group of scientists. To assess the cognitive, social, and economic aspects of different modeling approaches, we draw on a set of established theoretical lenses from different disciplinary backgrounds, ranging from cognitive linguistics to learning theories.

The structure of the paper is as follows. First, the subject of our investigation, multi-level modeling, is defined within the broader context of conceptual modeling. Then, we introduce a multi-perspective evaluation framework and use it to discuss different approaches. Subsequently, we point out that programming languages should be accounted for in order to exploit the full potential of multi-level modeling. Since the further development of multi-level modeling does not only depend on coping with remaining research challenges, next we outline possible paths of a dissemination strategy.

The paper concludes with a final discussion and an outlook on future research. Note that the paper targets both, the multi-level modeling community, and researchers not yet familiar with the characteristics of multi-level modeling and its promises. To address the latter, the following section presents an overview of essential concepts of multi-level modeling languages and explains why it can be regarded as a new paradigm.

2 The subject: essential characteristics and possible variations

The field of multi-level modeling comprises various approaches which are usually dedicated to the development of languages and tools, cf. e.g., [9,28,53,61,64,81,82,85]. Since we focus on multi-level modeling in general, notion specific approaches, we first need to clarify the subject of our investigation. On the one hand, this requires the identification

of common properties that can be regarded as characteristic for the field. On the other hand, it recommends accounting for the relevant context and the purposes of using multi-level modeling, because these factors are of pivotal relevance for the evaluation of languages and tools.

2.1 Core concepts

Multi-level modeling is a special kind of conceptual modeling. The term multi-level modeling covers any modeling approach that aims to provide systematic support for representing multiple classification levels within one model [8]. There are two families of multi-level modeling languages. Most languages fall in the object-oriented family, in the sense that they are based on notions of class and object, which are characteristic for object-oriented languages, e.g., [9,28,53,82,106]. The second family of multi-level languages is set- or logic-based, with some rather focusing on data or knowledge bases, e.g., [63,85,86,88], and others aiming at multi-level representations of ontologies [40]. While both families have much in common, they differ in at least one important aspect. While object-oriented approaches feature a concept of object, where an object is of one and only one class, logic-based approaches allow an object to be of many classes. Note that while the UML metamodel allows for an object to be of more than one class, object-oriented programming languages do not. Therefore, the prevalent interpretation of UML object models is that an object is of one class only. We do not entirely exclude logic-based approaches, but with respect to specific semantic aspects and the transformation into code, we focus on object-oriented approaches.

Within this focus, based on the analysis of publications, among others [9,28,53,61,64,81,82,85], corresponding discussions at MULTI 2017 and the Dagstuhl seminar on multi-level modeling in 2017, we identified core properties of multi-level modeling approaches that are described below. However, please note that the concepts which characterize multi-level models are similar to, but clearly different from corresponding concepts used in traditional object-oriented modeling. Therefore, applying traditional terminology to multi-level models may be inappropriate, and even misleading. Also, so far there is no unified terminology for multi-level modeling, cf. Sect. 6. Therefore, in order to prevent misunderstanding, we introduce the terms that we use in this paper together with presenting the core characteristics of multi-level modeling. The following description is supplemented with references to an example diagram created with the Flexible Meta Modeling and Execution Language (FMML^x) [49,53] (see Fig. 8). Choosing the FMML^x for this purpose is mainly related to the fact that it is suited to illustrate the core concepts and that it builds the foundation for tools we will use later to demonstrate the integration of modeling and coding. Note that the diagram also shows additional concepts (see Sect. 2.2). The corresponding labels are printed in italics.

An arbitrary number of levels Different from the metaobject facility (MOF), the representation of classes is not limited by a maximum number of classification levels. Note that the term "classification" does not have exactly the same meaning as in traditional object-oriented approaches. This is the case for the associated term "level" as well, cf. [72]. Therefore, we use the following terms. Through "concretization,"[2] in contrast to "instantiation," a "concretion," in contrast to "instance," of a class C on level m is created, which is a class on level $m-1$. Concretization requires that at least one feature (attribute, as well as, in some cases, operation and association) of a class is instantiated in the concretized class. Other properties will usually be "inherited" (see "deferred instantiation"). If all properties of a class are immediately instantiated, concretization is, at first, equivalent to instantiation. However, through adding further properties to the concretized class, it can be different from instantiation in that case, too. In the opposite direction, we speak of "intrinsic classification" in contrast to "classification" and "intrinsically classified" in contrast to "classified." Note that the traditional terms are still needed for those cases where they apply: Both instantiation and classification in a strict sense are possible in multi-level modeling, too. We refer to the tree of classes that are concretized from a class and its concretizations as a "concretization subtree." The excerpt of a multi-level model shown in Fig. 8 includes objects on five different levels. As the example illustrates, higher-level classes allow for the specification of knowledge that applies to a wide range of more specific classes—and that cannot be represented through generalization alone (see also the example in Fig. 1).

Every class is characterized by an explicit or implicit level There are different approaches in use to define explicit levels. Some approaches assign a level directly to a class [8,53]. While levels are usually expressed by numbers (either starting at the bottom or at the top), they may also be represented by terms that are supposed to clearly symbolize the intended level of classification [85] (cf. diagram in Fig. 2). Atkinson and Kühne distinguish between "ontological" and "linguistic" levels of classes [9]. While we regard a discussion of this distinction as important, we will not go into this discussion, because it goes beyond the core characteristics of multi-level modeling. The example diagram in Fig. 2 that was created with the FMML^x [53] features explicit definitions of levels. Levels are indicated by the background color of the field used for displaying the name of a class and a number in the same field. Other approaches favor an implicit definition of classification levels, cf. [11,61]. In that case, a classification level

[2] Adopted from [85,86], however, not with exactly the same meaning.

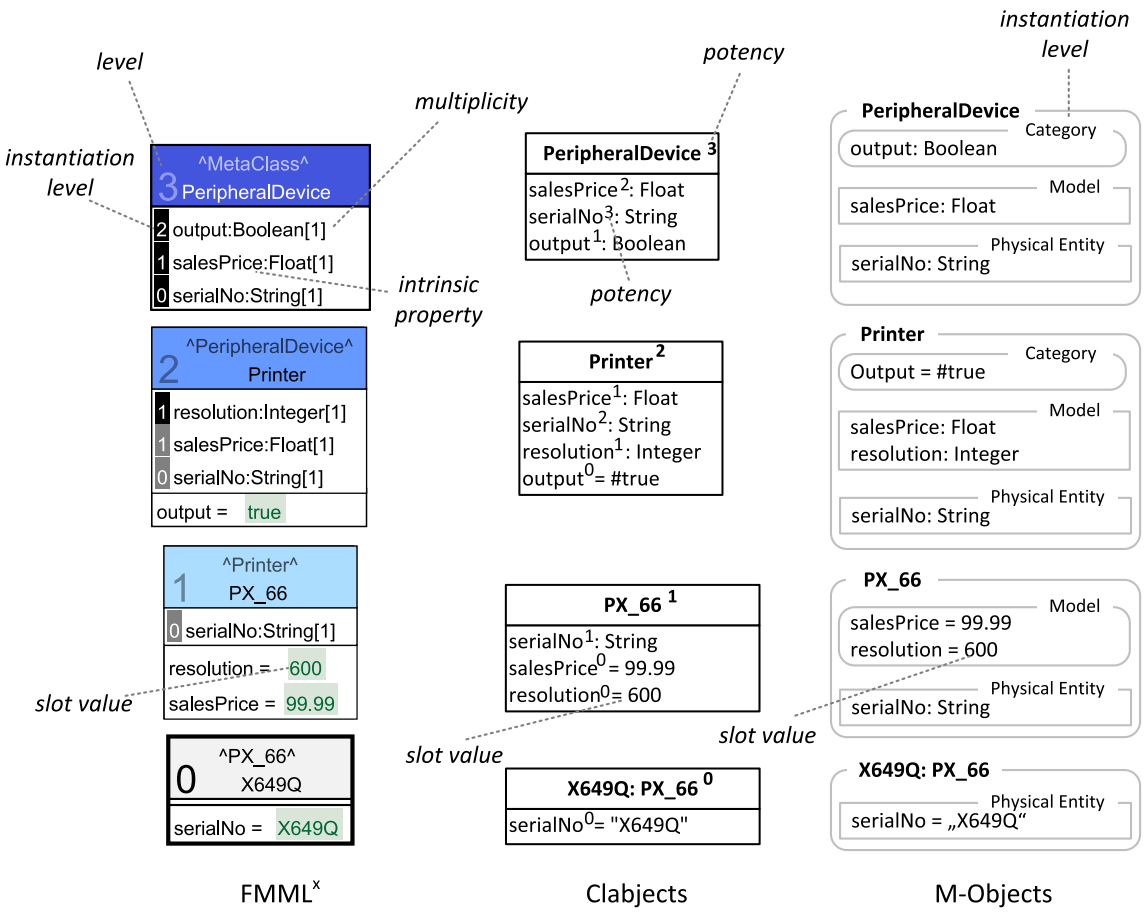

Fig. 2 Multi-level concepts presented with different notations

can be determined dynamically depending on the number of instantiation steps that can be performed until level 0 is reached (cf. clabject diagram in Fig. 2). Like in traditional object-oriented models, the level of an object that cannot be further instantiated is 0. Classes on any level n can be intrinsically classified to a class on level $n + 1$.

Every class is an object It is essential for classes in multi-level models to allow for having a state: Every class is instantiated (in the terminology proposed in this paper: "concretized") from a metaclass. That requires the instantiation of at least one property defined with the metaclass, and hence, a state of the instantiated class. Supplementing classes with the ability to execute operations is definitely useful as the example of the operation **`noOfModels`** in Fig. 8 illustrates. It is defined in the class **`PeripheralDevice`** and executed in the class **`Printer`** where it produces the result that corresponds to the number of its instances. Note that there are multi-level modeling languages that do not support the execution of operations.

Deferred instantiation The instantiation of an attribute defined with a class on a level $m > 1$ can be deferred to a level $l < m - 1$. This serves the expression of knowledge on a higher level n, even though it may be concretized only on levels below $n - 1$. For this purpose, Atkinson and Kühne [8] propose the use of a concept called "potency" which defines the number of possible instantiation steps of a class or an attribute. In turn, the FMML^x uses the term "intrinsic feature" to mark attributes the instantiation of which should be deferred, and an explicit instantiation level. Note that "intrinsic" should not be confused with its use in work on foundational ontologies, where it basically serves the distinction of attributes and associations (cf. [59,107]). "M-Objects" [85] use a similar approach, only that levels are not marked by numbers, but by terms. In this paper, we use the terms "intrinsic feature" and "instantiation level" as defined in [53].

The example in Fig. 1 served us to illustrate limitations of the traditional paradigm to express existing knowledge appropriately. In contrast, multi-level modeling enables to clearly express the intended meaning, which is illustrated by the diagrams shown in Fig. 2. The diagrams represent three widely equivalent multi-level models that were created with three different languages, namely the FMML^x, clabjects, and M-Objects. The class **`PeripheralDevice`**, which is an object at the same time, is located on level 3. While one of its

attributes (**output**) is instantiated within **Printer**, that is, on the level below, the instantiation of its other attributes is deferred to L1 and L0, respectively. Furthermore, it is possible to extend a class with further attributes, which is the case for the class **Printer**. Note that we use the FMML^x notation for the remaining figures in the paper.

While *associations* are important in multi-level modeling, too, there is no unified conception of associations, which would be part of the common core. Some approaches allow for associations between classes on different classification levels, e.g., [49,53], while others restrict the use of associations to classes of the same level only, cf. [4,79].

2.2 Additional concepts

There are various additional concepts that focus on specific aspects. The following extensions are not part of the common conception of multi-level modeling. Nevertheless, the additional concepts deserve a separate treatment since they directly relate to the idea of distinguishing multiple levels. More importantly, in our experience these additional concepts are often required for multi-level models of domains that are characterized by a distinct conceptual diversity. A more elaborate description of these and other extensions is given in [49].

Deferred instantiation of associations Some approaches allow for deferred instantiation of associations. While this corresponds to deferred instantiation of attributes, it is different from it. This is because the classes the association applies to may not be known at the level where the association is specified. This lack of knowledge may include multiplicities. The diagram in Fig. 8 includes the intrinsic association **compatible**, because it is already known at a higher level of (intrinsic) classification that peripheral devices and computers may be compatible. However, the association must not be instantiated before level 1.

Deferred instantiation of operations Similarly to attributes, the instantiation of operations that were defined with a class on level m can be deferred to a level lower than $m-1$. Figure 8 shows a corresponding example.

Contingent level classes There is a good reason for specifying the level of a class. With respect to the semantics of a class, it makes a clear difference whether it is meant as a class on level 1 or on a higher level. In natural language, we can cope with the ambiguity of terms like "Product," which can be used to refer to a particular product, a type of product or even to the set of all kinds of product types. In conceptual modeling, it is preferable to avoid ambiguity. Nevertheless, there are cases were the level of a class is not invariant across all possible use cases. In one context, it may be n, in another context is may be m, with $m \neq n$. Sometimes, the design of two multi-level hierarchies of similar objects results in a top-level metaclass C that makes sense for both hierarchies. But it may turn out that one hierarchy would require a level from C that is different from that required by the other hierarchy. There are basically two ways to cope with this kind of diversity. One could split a model in two or more models, where the class is on a specific level in each model. Or one could define the level of the class as contingent and, hence, allow for it being instantiated into classes on different levels. The latter corresponds to "skipping the instantiation" at the intermediate metalevels, as suggested by [30]. The introduction of contingent level classes creates a challenge to model integrity, because it would allow for two contradictory propositions: Class c is on level m (because one of its concretions is on level $m-1$) and class c is not on level m (because another concretions is on a different level) . One approach to cope with this problem is to refer to modal logic (for an overview, cf. [55]). Then, every specific level of a contingent level class would be part of one possible world [43] (see also Sect. 3.1.1).

Contingent instantiation levels of attributes The fact that a contingent level class should allow for being concretized into classes on different levels may have an impact on the deferred instantiation of its attributes. If, for example, the contingent level class "Product" can be concretized into "Automobile" on level 3 or, within another context with a less elaborate terminology, into "UsedCar" on level 1. In the first case, an attribute like "price" would be instantiated on level 1, in the second case, on level 0.

2.3 The multi-faceted conception of multi-level modeling

Traditional approaches to conceptual modeling are characterized by a clear distinction of executable objects (usually on M0), models, and modeling languages, which is most prominently reflected in the language architecture proposed by the MOF. With multi-level modeling, this distinction is blurred. A model that features multiple levels of classification may include elements that correspond to traditional model elements on M1, as well as elements on (meta) language levels such as M2, M3, and above. While it is conceivable to hide upper levels in order to reduce complexity for certain groups of users (cf. Sect. 5), a comprehensive representation of a multi-level model offers clear benefits to experienced users. It allows for the introspection of a model by navigating to classes on higher levels, that is, in the traditional sense: to specifications of DSMLs, and adapt these if required. The multi-level model in Fig. 8 in Sect. 5.2 illustrates how various levels of intrinsic classification are combined into one model or, in other words: how a DSML can be specified with a more general DSML. For example, a more generic language for modeling computer hardware could comprise a concept such as "computer" which could be used to model more specific concepts, such as "laptop," on the level below,

while these more specific concepts could then be used to model even more specific concepts. The generic concrete syntax provided by a multi-level modeling language like the FMML^x can be replaced by domain-specific notations. To this end, it is also possible to reuse and refine the concrete syntax defined for concepts on a higher level on lower levels. The graphical notation depicted in Fig. 8 indicates how classes that serve the specification of notation elements could be concretized to classes that refine the more general notation. A corresponding multi-level tool supporting such a refinement of notation elements is presented in [5].

The multi-faceted conception of multi-level modeling is of pivotal relevance for the purpose of our investigation. On the one hand, it suggests that neither those requirements proposed for the evaluation of conceptual models, e.g., [37,51,99], nor those proposed for modeling languages, cf. [46,94], reflect exactly what could be expected from multi-level modeling, cf. also [7]. On the other hand, related to that, multi-level modeling is likely to serve different groups of potential users. Those may include people who are interested in objects on level 0—and how they relate to concepts on higher levels. Different from traditional approaches, multi-level modeling allows for representing objects on level 0 together with classes/objects on higher levels.

This brief discussion already hints at additional challenges that come with multi-level modeling, compared to traditional modeling approaches. In the next section, we explore such differences systematically, thus setting the stage for the development of dissemination strategies.

3 Comparative evaluation

Since our focus is on object-oriented multi-level modeling, it is natural to compare it to traditional object-oriented modeling. At first glance, it may appear that such a comparison can be done easily. Since multi-level languages usually include traditional object-oriented modeling as an integral part and extend it with further concepts, they should be more powerful. While this is a valid argument, it is not necessarily convincing for two reasons. First, the additional concepts increase language complexity, which leads one to reconsider usability and comprehensibility. Second, an elaborate evaluation of a modeling approach should not be restricted to the concepts it provides, but also account for other aspects that influence its utility.

Due to its multi-faceted nature, we compare multi-level modeling also to traditional DSMLs, that is, those DSMLs that are based on a MOF-like language architecture. The motivation for a DSML is similar to that of multi-level modeling in the sense that both aim at providing higher-level language concepts. At the same time, they offer clear advantages over GPMLs. Therefore, it would be inappropriate to restrict the comparison to traditional object-oriented GPMLs only. Note that DSMLs may be specified within a multi-level language architecture, too. However, in that case, they would qualify as multi-level modeling approaches.

3.1 A multi-perspective framework

Conceptual modeling serves different stakeholders for a variety of specific purposes. Therefore, the evaluation of a modeling approach should account for different perspectives. The framework we propose comprises five perspectives (cf. Table 1). It is intended to cover most relevant criteria, but we do not claim it is complete. There are in fact two relevant aspects that are either not accounted for or to a limited degree only. Formal language properties such as correctness and completeness are relevant criteria to evaluate particular language specifications. However, they are hardly applicable to multi-level modeling in general, since semantics of specific multi-level languages differ in certain aspects. Therefore, formal criteria are not accounted for in the framework. Second, the concrete syntax, even though it may be of considerable relevance for the usability of a modeling language, is not directly accounted for either. This is for two reasons. First, the variety of specific notations of multi-level modeling languages does not allow for a general assessment—which is what we aim at. Second, a multi-level model may comprise one or more DSMLs on different levels of abstraction. These DSMLs may be added or modified along with their notations. Hence, there is not just one concrete syntax as it is the case, e.g., for the UML. The examples in Fig. 8 illustrate this multi-faceted nature of notations in multi-level modeling.

3.1.1 The essential design perspective: core abstractions

Concepts that allow for expressing abstraction are of pivotal relevance for developing models that serve as a foundation for adaptable and sustainable systems. To assess the utility of an abstraction concept, two aspects can be distinguished. First, what are common properties of what range of objects, a concept allows to represent in a single abstraction ("abstraction on"). Generalization, for example, allows capturing common static and functional properties shared by a number of classes within a superclass. This aspect indicates the potential for reducing conceptual redundancy, which in turn contributes to the ease and integrity of system changes (see, however, also Sect. 3.1.2) and fosters the reuse of more abstract artifacts in different contexts. The quality of an abstraction in this sense also depends on the integrity of modification. In that respect it makes a difference, e.g., whether an extension is monotonic (that is, it does not produce side effects on the extended parts, like, e.g., specialization) or not (like, e.g., extending a process model by inserting an additional activity). The second aspect of an abstraction concept relates to

Table 1 A multi-perspective framework—overview

Perspective	Aspects	Operationalization
Design perspective	Abstraction on and from	What are common properties of what range of objects, a concept allows to represent in a single abstraction? (*abstraction on*) What is its potential to fade out aspects that are deemed irrelevant for certain purposes? (*abstraction from*)
Engineering perspective	Reuse, integrity, adaptability	What concepts and mechanisms does an approach provide to mitigate conflicts that arise from focusing on these aspects?
Development perspective	Mapping to implementation level documents	Is efficient and consistent transformation of models into implementation documents supported?
User perspective	Cognitive fit	Do the language concepts clearly relate to natural language concepts? Are there semantic differences between modeling language concepts and corresponding natural language concepts that may contribute to misleading interpretations?
	Learning effort	What effort does it take to learn the adequate use of a modeling approach?
	Reduction of complexity	Does the application of the approach promise reduction of complexity?
	Customizability	Which options are available to empower users with the customization of modeling languages?
Economic perspective	Protection of investments	Are there any standards? Is the considered approach offered by a renowned vendor?
	Availability of trained professionals	To what extent are professionals available who are familiar with an approach? Is the given approach/paradigm covered by standard curricula?
	Availability of mature tools	What tools are available? Are they mature? Have they been used outside of the research context?

its potential to fade out aspects that are deemed irrelevant for certain purposes. In the ideal case, the aspects that are faded out may change over time without affecting the invariant core of a system ("abstraction from"). This aspect is especially relevant for the adaptability of a system. It is needless to say that the benefits enabled by these abstraction concepts are not compelling on their own. Instead, they chiefly depend on the quality of the required domain knowledge. If an abstraction used by a modeler is not invariant over time, it may seriously compromise the adaptability of a system.

Traditional object-oriented modeling The pivotal abstraction concepts provided by the UML are classification, generalization/specialization, encapsulation, composition/decomposition, and polymorphism. The corresponding assessment is shown in Table 2. Further abstraction concepts such as aggregation, composition, redefinition, or delegation are not accounted for, because they lack clear semantics and/or are not supported by every traditional object-oriented modeling approach.

Despite their undisputed expressive power, the abstraction concepts provided by traditional object-oriented languages are clearly limited. First, there is no way to account for further levels of classification. Second, related to the first, it is not possible to represent certain conceptual hierarchies appropriately. This well-known limitation is illustrated through the model in Fig. 1. It corresponds to the example in Fig. 8. Following the design guideline to represent knowledge about a domain on the highest level possible, the class **`PeripheralDevice`** includes attributes which in part suggest instantiation (**`input, output`**), in part inheritance, while others (highlighted with a grey background)—including an operation—seem to be neither inherited nor directly instantiated. The example also illustrates that the appropriate representation of knowledge may demand for classes that are objects at the same time.

DSMLs The abstraction concepts provided by DSMLs vary. They depend on the purpose of a language and particular design decisions. Any abstraction concept offered by object-oriented GPMLs could be offered by a DSML, too. There is, however, one important difference. The abstraction concepts of the object-oriented metalanguage a DSML is specified with apply to metaclasses and not, as in the case of a GPML, to classes. If corresponding concepts are to be provided by a DSML (for example, generalization/specialization), too, they need to be redefined in the corresponding metamodel, because it is not possible to simple reuse those concepts from the metalanguage: They are instantiated with the metamodel. In addition to object-oriented abstraction concepts, a DSML

Table 2 Assessment of object-oriented abstraction concepts

Concept	Abstraction on	Abstraction from	Comment
Classification (intentional)	Types of properties. *Advantage:* avoiding redundant specification; all instances of a class can be treated alike; additional property types that apply to all instances can be added—which may, however, require changing the state of instances	Specific properties; abstraction from variation of properties. *Advantage:* specific properties of instances can be changed without effecting a system's integrity	Can be changed, but not without affecting integrity of instances; deleting properties is likely to compromise (referential) integrity
Generalization/specialization	Common property types shared by a set of classes. *Advantage:* avoiding redundant specification, integrity; adding property types to superclasses immediately effective for subclasses	Specific property types of specialized classes; that is, from variant parts of a system. *Advantage:* specialization is monotonic, no side effects on general classes	Substitutability constraint needs to be accounted for
Encapsulation	Functions (with different degrees of protection/visibility). *Advantage:* unified external representation; prevention of unauthorized modification of object state	Data structure. *Advantage:* data structure can be changed without side effects	Only those changes of data structures that do not affect corresponding functions/operations have no side effects
Polymorphism	Abstract operation with uniform name. *Advantage:* improves readability of models/code through use of "natural" designators	Type of object that receives a message. *Advantage:* new classes with specific implementations of abstract operation can be added without side-effects on objects that send requests	Only possible, if operation interface is not affected
Composition/decomposition	Composite parts. *Advantage:* reduction of complexity by fading out details	The system in its totality (decomposition). *Advantage:* reduction of complexity through separation of concerns	Depends on specific semantics of composition

may introduce domain-specific abstraction concepts such as specific types of associations. The specification and use of DSMLs suffer from a problem that is similar to the one that limits the use of object-oriented GPMLs: It is not always possible to express all invariant knowledge about a domain in the language specification. Even if one definitely knows that instances of the classes that can be modeled with a DSML have certain properties, these cannot be expressed in the language specification.

Multi-level modeling Object-oriented multi-level models provide the abstraction concepts of traditional object-oriented modeling on every level, without the need to redefine them on lower levels. Abstracting a set of classes to a metaclass has the obvious advantage that the semantics of these classes is controlled by the metaclass, whereas specialization allows for arbitrary extensions. Different from a generic metaclass like "Class," a more specific metaclass is suited to clearly contribute to the intended integrity of the instances. Since metaclasses can be classified into meta metaclasses, this advantage applies to the construction of multi-level DSMLs, too. Different from generalization/specialization, metaclasses allow to specify the range of possible variations, which supports the integrity of respective system changes. The introduction of metaclasses allows to abstract from particular instances, which contributes to the reduction of model complexity and conceptual redundancy. If, for example, certain values that are mandatory for each instance, e.g., technical data of product types that apply to all of their instances, these values are assigned only once with the class.

In addition to the core concepts, extensions enable further abstraction. The additional concepts described in Sect. 2.2 allow for expressing contingent and incomplete knowledge. There are cases, where the level of a class may vary with the context it is used in. The concept of level contingency allows for abstracting on commonalities shared by particular interpretations on specific levels. To avoid conceptual inconsistencies and serious threats to system integrity, a contingent class can be interpreted similar to the possibility of a proposition in modal logic. The class being on level $l = n$ is true in one possible world, whereas it may be on a level m with $m <> n$ in another possible world. Allowing for deferred instantiation of associations as well as for the specification of attributes with classes on levels above 1 reflects the fact that the knowledge we have about lower levels is limited on

upper levels. For example, we know that a car, modeled as a class on level 3, has an engine. But we might not know the engine type that is assigned to a particular model on level 1. Nevertheless, it helps avoiding conceptual redundancy to abstract on all commonalities shared by cars on level 2 into the concept of car on level 3. It reduces the range of possible concretizations, and instantiations on L0, respectively. Thus, it contributes to system integrity. Incomplete knowledge can be expressed in traditional object-oriented modeling, too, e.g., through abstract classes. But, it is not possible to express incomplete knowledge that is later completed through instantiation. Contingent attribute classes and associations between contingent level classes allow for abstracting on contingent and on incomplete knowledge.

3.1.2 The engineering perspective: support for relaxing principal design conflicts

This perspective is very similar to the previous one, and it could as well be called design perspective, especially because it is also related to abstraction. However, it emphasizes a different approach. Instead of looking at concepts provided by languages, its focus is on principal design conflicts. They originate in conflicting measures to achieve generic goals that need to be accounted for whenever models, software, or languages are designed. This focus results in a specific assessment criterion: What concepts does an approach provide to mitigate principal design conflicts? The conflicts we explicitly account for are range of reuse vs. productivity of reuse, and adaptability vs. integrity. The weight given to them varies from project to project. But there will hardly be any doubt about their fundamental importance.

Reuse: range versus productivity: Ever since the legendary software engineering conference in Garmisch Partenkirchen in the fall of 1968, software reuse has been considered as the essential condition of building software that is both affordable and of high quality. On the one hand, reuse of existing artifacts is suited to clearly increase productivity of software development. On the other hand, economies of scale resulting from the range of applications an artifact can be reused in, contributes to lowering the price of an artifact. Unfortunately, both aspects are in an obvious conflict. The higher the level of semantics of an artifact is, that is, the more specific it is, the higher is its contribution to increasing productivity, but the lower the likelihood that it can be reused in a wide range of cases. With respect to design, various denominators exist for this conflict (cf. [19]), with the "power/generality trade-off" [89, p. 71 ff.] being among the most prominent. The conflict between range of reuse and productivity of reuse corresponds to a principle conflict not only in design, but also in theory building. While a theory should be generally applicable and of high expressive power, it is often the case that increasing one or the other is likely to lead to the falsification of a theory. Therefore, it is a common pattern in various disciplines to systematically constrain the scope of a theory to save it—which is a threat to its epistemological value. Therefore, Popper condemned this strategy and demanded that theories, on the contrary, should be designed in such a way that the are easy to falsify [96, p. 78]. While one may or may not follow Popper's advice for theory building, it is not a good idea to apply it to the design of artifacts that should satisfy certain requirements—unless we have no issues with not satisfying requirements. That means, we either have to decide for a trade-off that seems to be the least painful in a particular situation or try finding ways to mitigate the conflict. The following analysis is focused on the latter. Note that for the purpose of this analysis, we do not use semantics in the sense that there is one specific interpretation, but instead in the sense how many different interpretations it allows for, which is similar to the concept of information content as it is used in the theory of science, e.g., [96, p. 19]: The larger the number of interpretations that are excluded, the higher is the level of semantics defined for a construct.

Traditional object-oriented modeling: It corresponds directly to the idea of object-oriented GPMLs that they emphasize a wide range of reuse and widely neglect support for modeling productivity. Only models that are created with a GPML may put more emphasis on economies of scale or on promoting productivity. Every model specified with a GPML represents a certain trade-off between these two objectives, suffering either from a limited range of reuse or from a lack of support for specific use cases. The only way to relax this conflict is through generalization/specialization, which, however, is not always sufficient. The examples of classes specified with a GPML in Fig. 3 illustrate two shortcomings. First, the language concepts, such as "class" and "attribute," are so generic

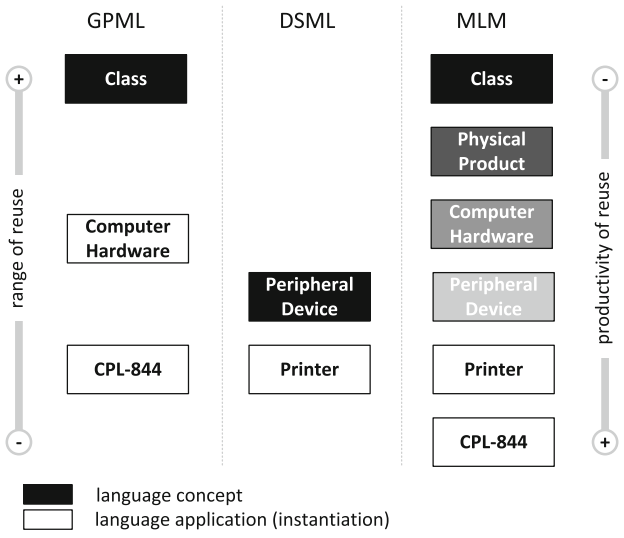

Fig. 3 Conflicting facets of reuse

that their reuse hardly promotes the productivity of modelers. Second, models created with a GPML can provide classes on any level of specificity, but each class defined with a GPML represents a particular trade-off. The conflict between range of reuse and productivity can be relaxed only through generalization/specialization, as long as that is an option. The use of GPMLs corresponds to a situation where people use a certain, more or less specific terminology. If somebody asks for the meaning of a term, the explanation would be something like "it is a thing." Accordingly, whenever a new term is introduced, it has to be defined from scratch using basic concepts such as "thing" and "property"—in all events where specialization is not appropriate.

DSMLs: Different from GPMLs, a DSML supports modelers with knowledge that may promote their productivity substantially. Also, the very existence of DSMLs provides evidence that specialization is not enough for the reuse of domain knowledge. However, similar to a GPML, every DSML represents a specific trade-off between range of reuse and productivity of reuse that can hardly be relaxed. If technical languages were defined like traditional DSMLs, people who speak the language would benefit from the embedded domain knowledge. The language could be used to define more specific terms, the explanation of which would refer to the language concept they were created with, e.g., "this is a peripheral device" instead of "this is a thing." However, that would comprise only one step of abstraction. If somebody continued asking, e.g., "what is a peripheral device," the answer would have to refer to basic language concepts—similar to GPMLs. Accordingly, the introduction of new concepts, e.g., "scanner," would require defining them from scratch—with the GPML that serves as the corresponding metalanguage.

Multi-level modeling: In contrast to GPMLs and DSMLs, multi-level models are not characterized by a clear separation of modeling language and model. Instead, every language concept can be specified with a higher-level language concept which in turn may be specified with a further, still domain-related concept. Therefore, multi-level modeling does not require a particular trade-off, but may take advantage of a wide range of reuse of its higher-level classes and promote productivity with more specific classes—which, at the same time, benefit from the economies of scale of higher level classes by re-using them. This corresponds to the evolution of technical languages. On a more general level, e.g., in a textbook, general concepts used in a certain field are introduced, like "computer hardware" in Fig. 3. The textbook terminology is then used to develop refined terminologies that fit more specific domains. Therefore, the design of a DSML in a multi-level language architecture does not have to start from scratch, but can reuse proven DSMLs with a wider scope.

Adaptability versus integrity: As long as one is not a follower of the slogan "embrace change" (expressed in the second principle of the "agile manifesto" [14]), it is a good idea to account for adaptability during system design already. That also suggests accounting for integrity, since changing a model or a schema of a system carries the risk of jeopardizing integrity. If a schema allows for a wider range of changes, that is, if it does not comprise constraints of some kind that exclude dangerous changes, adaptability is high, but integrity is threatened. Hence, there is a principal conflict between adaptability (in the sense of range of possible changes) and integrity.

Traditional object-oriented modeling: In the case of a general purpose modeling language, extensive changes to the schema are possible, that is, the range of adaptability is high. At the same time, the effort that is required for adapting the schema, as well as the related threat to integrity, is high, too. Any concept, even the most bizarre one, can be specified. Again, the only way to relax the conflict between adaptability and integrity is to make use of abstraction. Generalization/specialization allows for monotonic extensions of (super) classes. Hence, it supports changes that do not jeopardize the integrity of the corresponding superclasses. However, these extensions are not constrained through the superclass, which threatens the integrity of extensions.

DSMLs: If a DSML is used to specify a model, the chances to protect the integrity of the model during its modification are clearly higher than with a GPML, since semantics of a DSML constrains the range of possible modifications such as changing existing classes or the specification of new classes. At the same time, the range of adaptability is lower. It is intentionally restricted to those adaptations that are not in conflict with the domain knowledge embedded in the DSML. The particular trade-off between integrity and adaptability depends on the design of a DSML: The more specific it is, that is, the more semantics it represents, the lower is the range of possible adaptations it allows for, and the better is its contribution to integrity. A DSML that features a concept such as "computer hardware" requires more effort to specify the concept of a printer than a DSML that includes a concept like "peripheral device" already. It cannot, however, provide both a higher level of integrity and a higher level of adaptability simultaneously. Similar to a GMPL, a DSML may allow for using generalization/specialization to relax the conflict between integrity and adaptability.

Multi-level modeling: While both traditional DSMLs and traditional object-oriented GPMLs are based on the idea that a system rests on models (or a schema) on one level only, a multi-level architecture allows for the definition of a hierarchy of models (or DSMLs). Thereby, multi-level modeling provides additional measures to relax the conflict between adaptability and integrity, because it allows to combine the strengths of a rigid schema, namely to promote

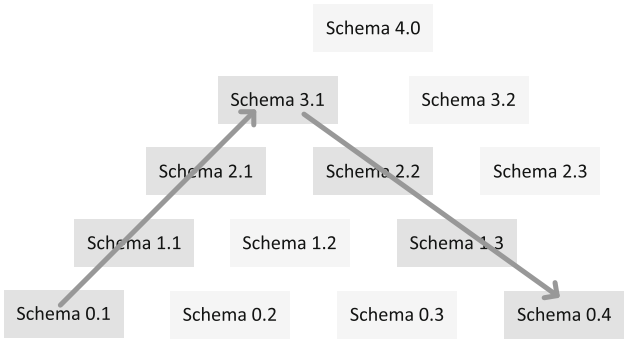

Fig. 4 Illustration of multi-level system architecture

system integrity, with controlled degrees of adaptability. The multi-level hierarchy shown in Fig. 3 allows for modifications on each level. If the particular types of printers covered by the (meta-) concept "Printer" are not sufficient for a certain purpose, a further concept of printer can be introduced on a higher level where its definition would be clearly more restricted than in the case of a traditional DSML—in the example by the domain-specific concept "Peripheral Device." Accordingly, and different from traditional system architectures, a multi-level system would not be restricted to one schema, but it could be based on a more or less extensive hierarchy of schemata on different levels. Figure 4 illustrates a multi-level system architecture. If the schema 0.1 needs to be adapted to new requirements and this cannot be achieved through monotonic extensions of the schema, the architecture would allow to navigate the schema hierarchy to a (meta) schema that allows for a concretization which fits the new requirements. The example shown in Fig. 4 indicates that one would go up to schema 3.1 and concretize it step-wise to schema 0.4. Note that the separation of different schemata serves illustration purposes. The schema hierarchy would more likely be represented in one multi-level model. That corresponds to an organization that combines the advantages of bureaucratic rules (consistency, reliability) with the strengths of flat hierarchies (agility, adaptability). It is needless to say that neither the construction of such a multi-level model nor schema migration is trivial. While modifications of schemata on higher levels are suited to enable convenient adaptations of an entire tree of schemata below them, changes that result in the deletion of concepts within a schema are a serious threat to integrity. The development of consistent change operations of this kind is a substantial research challenge (see [43] and comments on future research in Sect. 6).

3.1.3 The software development perspective: mapping to implementation documents

To support a seamless software development process, a conceptual modeling language should allow for the efficient and consistent transformation of models into implementation documents. This is not an inherent property of a modeling language, but rather reflects its relation to implementation languages. In the ideal case, modeling concepts correspond directly to concepts used in the respective implementation language. If that is not possible, there is need for unambiguous mappings from modeling concepts or model patterns to concepts of an implementation language. The mapping to implementation languages may not be possible without the loss of abstraction and, hence, information. As a consequence, the maintainability of code created through such a mapping would suffer. That is not a problem as long as code can be completely generated from models. However, this is not always the case. If, e.g., a modeling language allows for generalization/specialization and the implementation language, in contrast, does not, generalization hierarchies in the model have to be flattened in the code. Then, the abstraction mismatch between model and code will create an obstacle to the synchronization of a model after the respective code had been changed.

Traditional object-oriented modeling: There is an obvious correspondence between object-oriented modeling and object-oriented programming languages. Therefore, the mapping of models to code is well supported. However, as we shall see, this is not the case for the integration of models and code, which is required for using models during the entire life cycle of a system, cf. Sect. 4.1.

DSMLs: Mapping models created with a DSML to object-oriented code is not possible in a straightforward way. A DSML is usually specified with a metamodel, that is, through metaclasses. They define the semantics of the classes that can be modeled with the DSML. Since most object-oriented programming languages do not include metaclasses, a metaclass has to be mapped to a class, which could be instantiated only once into an object on M0 that would serve representing a particular class of the model. This loss of semantics does not only limit the utility of the code (particular instances cannot be managed as such). It also compromises the maintainability of code.

Multi-level modeling: The assessment of multi-level modeling leads to an ambivalent result. If multi-level models have to be mapped to traditional object-oriented programming languages, the result is even worse than with DSMLs, because more semantics is lost. The situation is, however, completely different, if a multi-level modeling language is supplemented with a multi-level programming language, cf. Sect. 4. In that case, a clear correspondence is enabled and the maintenance of code could benefit from the reduction of conceptual complexity enabled by multi-level modeling.

3.1.4 The user perspective

The users of a modeling approach comprise a wide range from domain experts who are not familiar with and not too much interested in modeling languages to seasoned system analysts and designers. Due to the diversity of prospective users, it is hardly possible to account for all of them explicitly. Instead, we distinguish two prototypes of users: occasional users such as domain experts with no or little modeling experience and expert users who have long standing experience with conceptual modeling. For both groups, a modeling approach should serve as an effective tool to structure and analyze a domain for a given purpose. While we assume that notation is relevant for users, we do not account for it, because that would require the analysis of particular languages. The following criteria are in part inspired by [45,47].

Cognitive fit: This criterion serves assessing how well concepts of a modeling language and the models created with it fit the cognitive structures of prospective users. Applying it suggests to analyze two interrelated aspects of conceptual modeling. First, the correspondence of language concepts to natural language provides an indication of the comprehensibility, both of modeling languages and models. Semantic differences between modeling language concepts and corresponding natural language concepts may contribute to misleading interpretations. This criterion is especially relevant for occasional and novice users. Second, it refers to principal patterns of human conceptualization, that is, how people develop and make sense of categories. This criterion is especially relevant for occasional and novice users with respect to comprehensibility and acceptance. It suggests to account for research in cognitive psychology and cognitive linguistics. In particular, cognitive linguistics, with its key tenet of natural language being reflective of how we think and act [74], is relevant in this respect. Note that this criterion is not meant to take natural language or even the inappropriate use of it as a model for the design of modeling languages. A plethora of studies in cognitive psychology show that humans struggle with logic and are often not able to provide consistent definitions of concepts they seem to be familiar with [34,34,68,109]. These cognitive limitations and the corresponding vagueness of natural language are a clear obstacle to designing consistent models, especially with respect to novice users. However, presenting users with language concepts that are in confusing contrast to concepts their cognition is based on carries the risk to produce confusion and would, hence, be counterproductive.

Traditional object-oriented modeling: It is a widespread assumption that object-oriented modeling provides intuitive concepts that correspond directly to the concepts humans use to structure and understand the world. The claim that an "object model" resembles "human cognition" [20, p. 2] or Meyer's famous exclamation "Many objects are just there for the picking" [84, p. 117], go hardly undisputed. While there is evidence that objects and categories of objects (concepts) are pivotal for human cognition, there is also evidence that the way humans build concepts can be clearly different from the construction of classes in object-oriented modeling. Different approaches in (cognitive) linguistics, e.g., idealized cognitive models [75, p. 91], prototype theory [97], "Gestalt" theory [83], family resemblance [76, p. 12], and schemas [104, p. 10], hold that people categorize a given instance relative to a prototype, rather than in terms of necessary and sufficient conditions (the latter being presumed by classical approaches to categorization [57, p. 249]). This means that, in deciding on category membership, people compare a given instance to an abstract mental representation—the prototype class—that best represents instances of a category [57, p. 249]. The category that is chosen for an object may vary with the context. This prototype-based view is in obvious contrast to logic and to object-oriented modeling, since it does not fit the idea that the relationship between an instance and its class is clearly defined (either intentionally or extensionally). Instead, cognitive linguistics suggests "membership gradience" [75, p. 12], i.e., the classification of an object can be perceived as more or less typical. But even those who are familiar with logic may struggle with object-oriented concepts, since the notion of a class (an object is of one and one class only) is different from that in logic, where an object can be of many classes.

What does that imply for the evaluation of traditional object-oriented modeling? First, the assumption that object-oriented concepts clearly correspond with natural language and human cognition seems too daring. This is especially relevant with respect to novice users. However, cognitive fit is clearly better with experienced users of object-oriented modeling languages. Many of them are likely to agree with Meyer. At the same time, they might well ask themselves whether they do not suffer from a *déformation professionelle*. That would be a problem, if it compromised the quality of the models. However, the empirical fact that humans use conceptualizations different from those suggested by object-oriented modeling does not necessarily disqualify the latter. Similarly, the fact that most people struggle with logic does not disqualify logic as a formidable tool for thinking. There is evidence that supports this viewpoint: Philosophical ontologies like those proposed by [23] or [58] show obvious similarities with object-oriented concepts. They do not come with the claim to represent categories that humans naturally prefer, but rather with the proposition that they are especially suited to develop a proper structure of the world. Therefore, the evaluation with respect to cognitive fit leads to an ambivalent result.

DSMLs: There is a large range of concepts that might be provided by DSMLs. Therefore, a general assessment of cognitive fit is not feasible. However, in principle, DSMLs allow for the specification of concepts that correspond directly to

the technical terminology users in a certain domain are familiar with. In addition, a descriptive concrete syntax is suited to strengthen this correspondence. There is, however, one drawback that compromises the "natural" use of DSMLs. The specification of a DSML requires a clear distinction of language and language application. Apart from the fact that there are hardly convincing criteria to guide this distinction (for a proposal of corresponding criteria see [45, p. 15 f.]), such a distinction is not common in natural language. Take, for example, a DSML to model appliances. It may include a concept like "Cooker," which may be used to model an induction cooker. In natural language, both would be regarded as concepts, and, hence, as belonging to a language.

Multi-level modeling: Multi-level modeling features similar basic concepts as traditional object-oriented modeling. Therefore, the assessment of cognitive fit corresponds with that of traditional object-oriented modeling. The additional abstraction concepts may lead to concerns that they do not fit human cognition. It is hardly presumptuous to assume that even educated people will struggle with classes on levels above 1 or with ideas such as deferred instantiation. That would suggest that the friction between multi-level models and human cognition is more distinct than in the case of object-oriented modeling.

However, research in cognitive linguistics and in cognitive psychology indicates that remarkable similarities between natural language and aspects of multi-level modeling exist. People use abstraction hierarchies to organize their knowledge. Structures used for conceptualization are referred to as frames [35], domains [77], or schemata [78]. In particular, work on cognitive grammar by Langacker [78, pp. 132–137] who introduced the notion of "schematicity" shows striking similarities to multi-level modeling. Schemata are organized in hierarchies, where a schema represents certain features that are regarded as characteristic. A schema can be refined into more specific schemata, which in turn may be refined into further schemata. It is important that schema refinement is not restricted to a certain kind, such as adding further features. It may also comprise deleting features or assigning more specific values, etc. [104, p. 4]. Hence, the creation of levels in multi-level modeling would be one specific case of schematicity.

Further evidence is provided by learning theories. In his studies of human learning strategies, Bateson distinguishes five evolutionary steps, where every step is characterized by an increase of abstraction, which broadens the range of possibilities [12, p. 283 f]. In a similar vein, studies in cognitive psychology have shown that major advances in human learning require once in a while to "reframe" our view of the world or, in other words, to go beyond existing conceptualizations [108, p. 95 f]. As a consequence, there does not have to be a substantial conceptual friction between a multi-level model and human cognition. This is, of course, a preliminary assumption that requires further refinement and examination. Apart from that, it seems appropriate to assume that the ability and willingness to use multi-level abstractions vary, which suggests to provide different groups of users with different levels of abstraction.

In addition, many taxonomies, e.g., in biology, support the claim that multi-level modeling is not in conflict with common use of natural language. Often, taxonomies feature abstraction hierarchies that do not exactly correspond to generalization/specialization, since they assign values to concepts on higher levels that apply to lower levels only. For example, the category "cat" is characterized by "number of legs = 4," which is valid for all subcategories such as "tiger" and "house cat," but literally applies only to specific exemplars.

Learning effort: The effort it takes to learn how to adequately use a modeling language is an ambivalent aspect, but needs to be accounted for. Modeling projects do not only involve expert users, but also participants who are in touch with models only occasionally and who are not able or not willing to spend much effort to learn a modeling language. In this sense, this aspect is directly related to cognitive fit.

Traditional object-oriented modeling: There are various studies that report how novice users struggle with learning GMPLs like the ERM or the UML (e.g., [13,101]). However, it is not entirely clear whether this is caused by idiosyncratic language features or by more general limitations concerning the proper use of logic and abstractions. In addition, mastering a modeling language is not restricted to learn the concepts it offers, but includes their appropriate application, which will usually be the bigger challenge.

DSMLs: DSMLs are suited to clearly decrease the learning effort for those who are familiar with the domain that a language targets. However, if language concepts are different from those offered by the respective domain terminology, learning may be especially demanding. That is likely the case, too, for those who are not familiar with the domain terminology.

Multi-level modeling: Assessing the learning effort related to multi-level modeling demands for a differentiated approach. On the one hand, learning the specific abstraction concepts, the specific notion of levels, deferred instantiation, etc., is challenging, because it adds complexity and it is in contrast to known traditional modeling languages. This is a challenge not only to novice, but also to expert modelers. However, it is not mandatory that users are confronted with this challenge. Multi-level modeling does not imply to learn the basic abstraction concepts. Instead, the use of a multi-level model may be restricted to levels that correspond to abstractions, users are familiar with. Figure 7 shows an example of such a diagram. It comprises classes/objects on M1 and on M0 only. Users who are familiar with the UML should not have a problem with making sense of the diagram (even though

it clearly extends the representations possible with regular UML class diagram editors). In addition, the representation of a multi-level model does not necessarily require to explicitly account for levels. The example in Fig. 10 shows how a multi-level model can be presented to users in a GUI that does not require the notion of levels, but includes only concepts, users of the corresponding domain should be able to understand.

Reduction of complexity: This criterion is relevant from the engineering and the economic perspective, too. It is assigned to the user perspective, since reduction of complexity should not only foster maintainability and increase modeling productivity, it should also be suited to reduce the cognitive load users have to cope with. On the one hand, this criterion relates to the abstraction concepts provided to modelers. On the other hand, it relates to the complexity of the language itself. Note that the reduction of complexity will usually imply to increase complexity at first. Therefore, a language may be more difficult to learn and it may be more demanding to apply it properly, but this additional complexity may pay off in the end by enabling more effective reduction of complexity. However, while it may make sense for professional modelers to take the extra effort to learn a more complex language, occasional users will likely be put off by this kind of complexity. Therefore, it should be possible to hide complexity from certain groups of users.

Traditional object-oriented modeling: As far as reduction of complexity is concerned, the assessment is ambivalent. On the one hand, the abstraction concepts considered above foster the reduction of complexity. On the other hand, a GPML forces modelers to develop a domain model from scratch instead of providing them with proven, domain-specific abstractions.

DSMLs: Compared to GMPLs, DSMLs are suited to clearly reduce complexity for language users, because they relieve them from the burden to structure a domain from scratch. The benefit provided by this reduction of complexity depends on how a DSML fits the domain it is used in.

Multi-level modeling: Again, the assessment of multi-level modeling depends on the use case. An elaborate multi-level model that also represents proven domain concepts will clearly promote the reduction of complexity, similar to DSMLs. Using a plain multi-level language only provides users with powerful measures to reduce complexity. However, their proper use may increase complexity at first.

Customizability: A modeling approach should provide means to adapt its concepts to specific needs. It is hardly possible to foresee all relevant use scenarios of a modeling approach. Safe and convenient means to customize a language clearly increase its utility. Customizability corresponds to adaptability, but is focused on those adaptations that can be done by users. For that purpose, they need to follow certain change patterns that allow for safe and fairly convenient adaptation. The options offered by the three compared approaches to enable this kind of adaptation are illustrated through the simplified language architectures in Fig. 5.

Traditional object-oriented modeling: Customizability of traditional object-oriented modeling languages is clearly limited. Extension mechanisms such as UML profiles are widely used and have shown their usefulness. However, they are restricted to monotonic extensions of the metamodel (that is, they do not allow for changing the UML itself) and to extensions or modifications of the concrete syntax. Modifications of language concepts, especially of domain-specific concepts, are not possible. First, they are out of the scope of a GPML. Second, the metamodel of a GPML is on M2. Therefore, it does not allow for the instantiation of language concepts on M2.

DSMLs: A DSML can be designed for customization. As shown in Fig. 5, the meta-metamodel on M3 may include attributes that allow specifying on M2 whether a language feature is mandatory or optional. If, e.g., an attribute like **`salesPrice`** was marked as optional on M2, the language user would have the choice to include it in his variant of the language or not. At the same time, it would be possible to prevent the deletion of properties that are marked as mandatory. Extending a DSML by adding further concepts, that is, metaclasses, is conceivable, but that is likely to require a non-trivial modification of the metamodel. Therefore, it hardly qualifies as a customization option. It is not possible to specify features that apply to objects on M0, though. Even if one knew that every particular product needs to carry a serial number, this knowledge could not be expressed with the language specification.

Multi-level modeling: A multi-level model enables powerful customization options, which result from additional levels of classification. Therefore, customization is not restricted to a particular language level, as it is the case for traditional DSML, but can be applied to any level. Customization on any level can be supported or restricted by higher-level language layers. If, for example, a higher-level class such as **`PeripheralDevice`** defines the optional intrinsic attribute **`serialNo`**, it could be activated or not with the customization of the class **`Printer`**. Apart from that, it is possible, like with traditional DSMLs, to define additional attributes that were not already specified on a higher level, like, e.g., **`resolution`**, because the meta concept **`Attribute`** is available on every level above L2 (see corresponding metamodels in [25,49]). Adding classes to a level is a valid customization option as long as a new class can be concretized from a (meta-) class on a higher level. If that is not possible, the required modifications are too demanding to leave them to users. Different from a traditional DSML, a multi-level model allows adding properties that are intended

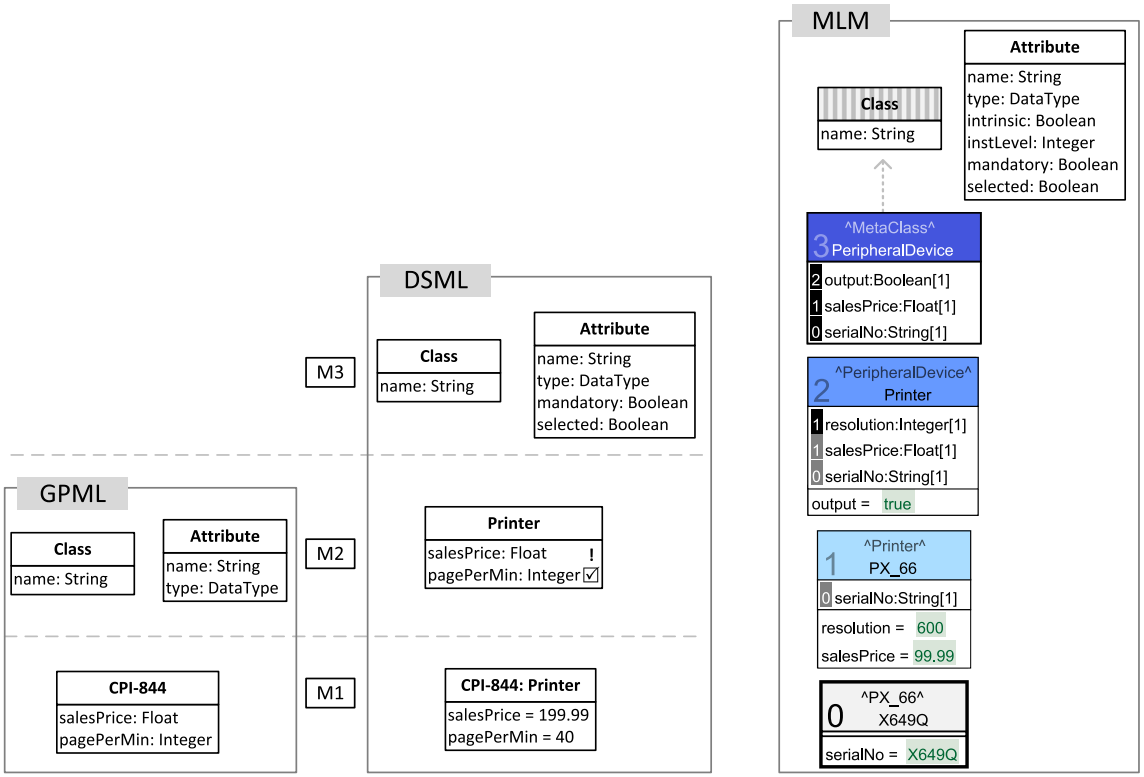

Fig. 5 Illustration of customization options

Table 3 Assessment of object-oriented abstraction concepts

Customization	Example	GPML	DSML	MLM
Add property accounted for in language spec	Attribute **`pagePerMin`**	Not an option	Possible	Possible
Add property not accounted for language spec	Attribute **`resolution`**	Not an option	Possible	Possible
Remove property	Attribute **`salesPrice`**	Not an option	Possible, could be prevented	Possible, could be prevented
Add new concept	Class **`Scanner`**	Not an option	Conceivable, but no support	Possible, at best through concretization of metaclass
Add property of instance on M0	Attribute **`serialNo`**	Not an option	Possible	Possible

for instantiation on M0 only, such as the intrinsic attribute **`serialNo`** in Fig. 5.

Table 3 summarizes the comparative assessment of customization options.

3.1.5 The economic perspective: does it pay off?

For a modeling approach to be successful in practice, it should promise economic benefits over other approaches. Some of the criteria discussed above are of obvious economic relevance, such as contribution to modeler productivity and model quality or user acceptance. In addition, there are a few specific aspects that are widely orthogonal to inherent properties of a modeling approach.

Protection of investments: The investments into a modeling approach can be substantial. Therefore, their effective protection is a relevant issue. It is promoted by reliable vendors and standards.

Availability of trained professionals: Modeling is an ambitious activity that demands for highly skilled people. Therefore, the availability of trained professionals, either in-house or with external service providers, is a mandatory prerequisite. At the same time, it has an impact on costs.
Availability of mature tools: The creation and use of conceptual models will usually require modeling tools. These tools have to feature a certain degree of maturity. The larger the market and the higher the competition between tool vendors is, the better will the terms and conditions for acquiring tools be.

Different from the previous perspective, the three criteria that form the economic perspective are not treated separately, since they are closely related.
Traditional object-oriented modeling: At first sight, it seems obvious that object-oriented modeling in general and the UML in particular promote economics of conceptual modeling. Since the UML is not only standardized by the OMG, but also widely used in industry, it is suited to protect investments into tools and models. In addition, the wide dissemination of the UML has promoted the availability of professionals who are familiar with the language. It also enabled the evolution of a market for tools which are available at low costs. However, as far as productivity is concerned, GPMLs provide little support.
DSMLs: With respect to those economic aspects that do not relate to inherent language properties, DSMLs will in general perform worse than GPMLs. On the other hand, their contribution to productivity should be clearly better. The productivity gain enabled by DSMLs is supported by various, sometimes enthusiastic reports on the benefits they provided in particular use cases, cf., e.g., [70,105]. These two opposite aspects of DSML economics recommend to evaluate the trade-off between a DSML and a GPML in each individual case.
Multi-level modeling: As far the availability of tools and professionals is concerned, the economics of multi-level modeling is disadvantageous, to say the least. This is the case, too, for the protection of investments into tools. However, the investments into training may even pay off, if multi-level tools are not available, since it is suited to develop modeling competence in general. The contribution to productivity depends on the availability of multi-level models that can be reused and adapted to specific needs. If a variety of domain-specific multi-level languages/models exist, model productivity would benefit even more than with corresponding traditional DSMLs, since it would be possible to use a higher-level DSML to specify a more specific DSML. Similar to DSMLs, the ambivalent economics of multi-level modeling have to be evaluated in each individual case.

3.1.6 Resumé

The comparative evaluation of multi-level modeling confirms its strengths in terms of enabling more abstraction. In particular, it does allow not only to combine specific benefits of DSMLs and GPMLs, but to relax essential design conflicts to a degree that clearly exceeds the possibilities of both. Similarly to the design conflicts discussed above, multi-level modeling allows relaxing the conflict between open and efficient communication, or, in other words, the conflict between a high level of integration ("tight coupling") and a low level of integration ("loose coupling"); in cases, where a class is too specific for a participating component, it is still possible to use a higher-level class as a reference to enable communication. If, e.g., a component receives an object of the class "CPL-844," which it does not know, it could still ask the object for its class and apply a useful interpretation as long as it knows that class, e.g., "Printer." That would be clearly better than getting an answer like "I am of Class" as it would be given in traditional object-oriented systems.

Furthermore, the concern that multi-level modeling is in contrast to conceptualizations users are familiar with must be put into perspective. While the complexity represented by some multi-level models may overwhelm many, there is, at the same time, evidence that multi-level models correspond to the construction of concepts in natural language, especially in terms of a correspondence to concept hierarchies discovered in cognitive linguistics. Against this background, it seems clearly more "natural" to allow for an unlimited number of levels than to introduce an arbitrary upper limit as it is the case for traditional object-oriented modeling and traditional DSMLs.

These strengths and the potential of multi-level modeling are contrasted by three serious drawbacks. First, from an economic perspective, multi-level modeling suffers from a lack of unity, mature tools and dissemination, which creates a serious obstacle to its use in practice. Therefore, for promoting the future development of multi-level modeling, its dissemination is of pivotal relevance. Second, only a few preliminary approaches exist that provide specific support for the design of multi-level models [31,48]. This lack of specific multi-level design methods is a serious restriction, too, since most prospective users are likely overwhelmed with the construction of multi-level models, cf. also [66], which results in two recommendations. On the one hand, there is need for specific multi-level modeling methods; on the other hand, tools are required that allow taking advantage of multi-level models, but include means to hide their complexity from users. Third, the utility of multi-level models is substantially limited as long as their implementation is restricted to traditional object-oriented languages. That recommends widen-

ing the scope to include multi-level programming languages, too.

4 Extending the scope: why programming languages should be accounted for, too

Even though conceptual modeling is not restricted to the design of software systems, the support of software development is arguably its pivotal purpose. Therefore, the transition from model to code should be as smooth as possible. That requires accounting for programming languages, too. A prominent example of this transition is the generation of code from models, which is at the core of most approaches to model-driven software development (MDSD).

4.1 The (avoidable) pain of synchronization

MDSD is based on two plausible assumptions. First, conceptual models are a more suitable medium to analyze and design a system than code. They enable a higher level of abstraction, and, especially in the case of DSMLs, foster communication between different stakeholders and promote the productivity of software designers. Second, coding is a cumbersome and error-prone activity. These are convincing arguments, which are supported by various studies that confirm a clear gain in productivity, especially with the use of DSMLs, e.g., [41,69,70,103]. In addition, specific architectures for implementing MDSD on a tool stack have been proposed, e.g., [93], in order to protect investments by abstracting implementation languages and run-time platforms away. Despite the plausible benefits of MDSD, it has a limitation resulting simply from the fact that models and programs are represented in different documents, which has two implications. First, there is a need to synchronize both representations after changes were performed on one document. As long as synchronization cannot be completely automated, which is rather the rule than the exception, the separation of representations is a clear threat to integrity and also to the investments into models, as they are likely to be outdated over time. Second, due to the lack of integration of both representations, it is not possible to inspect additional information provided by models during the execution of code. In the ideal case, reflection would not only allow for introspection, that is, getting information about the (meta) model, which defines the conceptual foundation of the code, but also to change the foundation and the code simultaneously. Research on "models at run time" is aimed at enabling this kind of reflection, sometimes extended by reasoning on models [3,17]. That includes work on control mechanisms that enable a software system to adapt itself to new requirements at run time, which are represented in corresponding models, e.g., in goal models [24]. However, these mechanisms are restricted to certain predefined ranges of adapting code to changes in corresponding models. They are not intended to support mutual adaptation or synchronization.

There are approaches that aim at closing the semantic gap between programming and modeling languages through a common metamodel. UMPLE [80] provides a common foundation for UML and Java. To this end, it extends Java with concepts to represent associations. Thus, it allows storing a UML class diagram as a Java program (except for the representation of the diagram layout). "Java Model Parser and Printer" (JaMoPP) [39] is taking a similar approach, mainly for reengineering purposes. Like UMPLE, it allows treating a Java program as an instance of the metamodel and, hence, as a model that could also be presented using a graphical notation. The model could then be edited in a graphical editor and stored again as Java code (together with additional information on the layout of the diagram). MoDisco [21] is a further example of this approach. While these tools allow for a tight integration of models and code on a conceptual level, they still require the synchronization of models and code. Independent from that, the approach to instantiate models and code from a common metamodel does not work for multi-level modeling unless multi-level programming languages are available. Generating code created with a traditional object-oriented programming language from models with multiple levels will always result in a loss of information and conceptual redundancy within the code. In other words, the "accidental complexity" that could be avoided through multi-level modeling is then embedded in the code. As long as code cannot be entirely generated from a multi-level model, that would substantially compromise software maintainability. Independent from that, "flattening" a multi-level model into traditional object-oriented code would not allow to inspect the intended conceptual foundation of software at run time.

At this point, we refrain from considering further approaches to synchronize models and code. Instead, we focus on a principal question that relates to the integration of models and code and, hence, to both, synchronization and introspection: Why is there a need to generate code anyway?

4.2 Prospects of a common representation of models and programs

Even those approaches that feature a common metamodel of code and models and that allow to represent models through code depend on the separate representation of models and code during the time a model is being edited in a graphical model editor. This is for a principle limitation of traditional object-oriented programming languages. Assume a class diagram is edited with one of these tools. That requires classes to be represented as objects on M0 in the graphical model editor. In traditional languages (with one level of objects only), there is no other way. It is not possible to represent a class that is part of a model with a class in the model editor, since

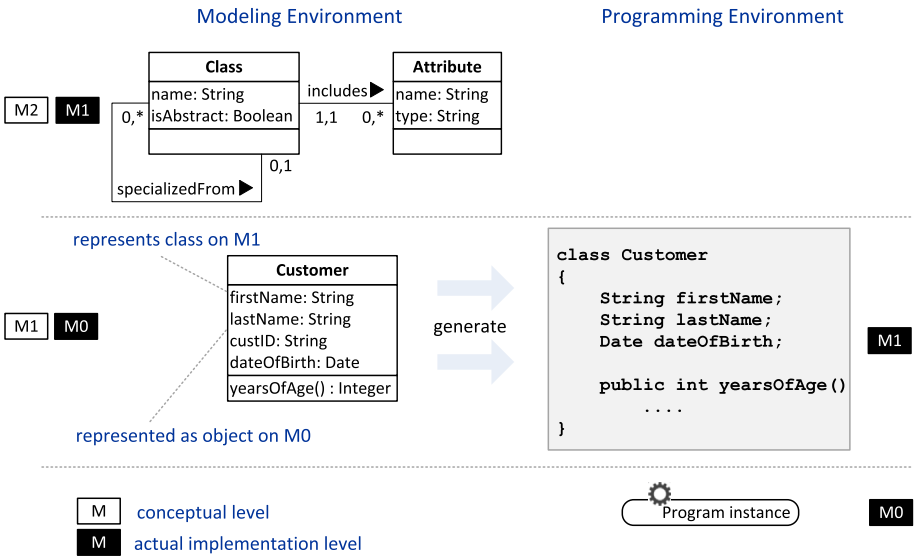

Fig. 6 Illustration of abstraction mismatch between models and code in traditional language architectures

that would require that classes may have a state to represent properties or references to other classes. Therefore, the model cannot be executed, since that would require instantiating it—which is not possible because it is represented through objects on M0 (see illustration in Fig. 6) and not on M1, where it conceptually belongs.

Integrating a multi-level modeling language with a multi-level programming language would solve two problems at the same time. First, there would be no semantic mismatch between a multi-level model and the corresponding code. Second, it would be possible to enable the common representation of models and programs during run time. If a multi-level modeling environment is implemented with a multi-level programming language, the classes (or metaclasses) that represent a model will be represented as objects on the same level in the editor. Therefore, a multi-level programming language would allow for the integration of modeling environments with run-time environments. It would enable to instantiate (or concretize, respectively) classes at any level from their metaclasses within the model editor, thus resulting in a substantial gain when it comes to run-time adaptability. Furthermore, it would be possible to provide users with multi-level reflection. Smalltalk users are familiar with the benefit of navigating not only through objects and classes, but to also inspect metaclasses at run time. However, Smalltalk is restricted to one metalevel only and to metaclasses that must not have more than one default instance.

There are a few multi-level programming languages. *DeepJava* [73] and *Deep Ruby* [87] extend object-oriented programming languages with the capability to create classes at run time and to enable an arbitrary number of levels. *XMF* [26] is based on a dedicated, reflective language architecture, the core of which is defined in a compact, recursive meta model, *XCore*. The architecture enables the construction of classes on an arbitrary number of levels. The levels are not defined explicitly, but instead determined dynamically. The *XModeler*, the language engineering environment that XMF is part of, serves the specification and implementation of programming and modeling languages as well as the execution of corresponding programs and models. It also features a metamodeling editor that supports the generation of corresponding model editors from meta models. The already mentioned FMML^{x}, a multi-level modeling and execution language, is implemented as an instantiation of an extended version of XCore. The corresponding version of the XModeler, XModeler^{ML}[3] (see screenshot in Fig. 9), features a common representation of multi-level models and multi-level programs at run time. Code and model diagrams are two (out of many possible) views on this common representation. This enables system architectures that clearly contribute to empowering users (cf. the outline of self-referential enterprise systems in Sect. 5.4): The model that represents the code can be navigated and possibly changed at run time. A further approach to enhance the attractiveness of multi-level modeling is represented by tools that allow for model analysis, retrieval and reasoning such as, e.g., by *Melanee* [6] or *ConceptBase* [62].

5 Dissemination strategies

No matter how attractive an artifact looks from an academic perspective, its utility will usually depend on its dissemination. This is especially the case for languages and their

[3] The XModeler^{ML} as well as various example models can be downloaded from the webpages of the project "Language Engineering for Multi-Level Modeling" at https://www.wi-inf.uni-duisburg-essen.de/LE4MM/.

corresponding tools. It does not require an extensive analysis of the current state to realize that the lack of common recognition and of noteworthy dissemination indicate a considerable weakness. It is not only a problem for those who regard the widespread use of a language its primary purpose and the ultimate proof of its utility. It is unsatisfactory, too, for those who mainly aim at advancing the state of research. The consolidation of the state of the art and the development of tools require resources that are easily beyond the possibilities of a relatively small group of researchers, most of whom cannot even dedicate all their resources to research on multi-level modeling. Furthermore, the remaining research challenges, cf. Sect. 6, are substantial, which also demands for a larger community of dedicated researchers.

In general, prospective users are hard to convince of the advantages of a new approach. Even if that can be accomplished, most users will still be reluctant to adopt it. On the one hand, they do not want to sacrifice their previous to artifacts designed with traditional languages. On the other hand, they might doubt that investments in new languages are sufficiently protected as long as they are not widely disseminated or even standardized. There are basically two options to overcome this dilemma. If an approach has the disruptive power to substantially improve the economics of using modeling languages (increase productivity, cut costs, etc.), there are good chances that more and more users will adopt it. If that is not likely to happen, the second option would be to apply a less offensive strategy, by introducing the new approach as an add-on to traditional approaches. With regard to the multi-faceted conception of multi-level modeling, the choice of a dissemination strategy depends on specific purposes and characteristics of prospective users groups. What may serve as a convincing argument to consider multi-level modeling as an alternative for some, may be not relevant to others. Also, some prospective users might not be interested in a comprehensive analysis of prospects offered by multi-level modeling. Therefore, in addition to providing convincing explanations of specific incentives, narratives might be helpful in arousing interest in multi-level modeling [22].

In the following, we distinguish between different purposes and user groups to outline four prototypical dissemination strategies. They all rely on the integration of a multi-level modeling language with a multi-level programming language. Note that the idealized target groups referred to in the strategies do not necessarily represent actual groups. Instead, they serve as prototypes. By its very nature, the outline of strategies is in part speculative and not free of contingencies. Therefore, it is mainly intended to contribute to a needed debate on dissemination.

5.1 Hiding those additional levels or: intrusion through the back door

The primary target group of this strategy are system analysts and teachers who regard traditional modeling languages such as the UML as a commodity. They are not too much interested in peculiarities of language semantics. Their focus is more on the pragmatics of modeling. They know that involving users or teaching students can be very challenging, mainly because people often struggle with the level of abstraction imposed by conceptual models. Accounting for exemplary instances and allowing for interaction with a model are suited to induce attention and understanding.

The incentives that are provided with the use of a multi-level modeling tool are obvious. Investments in previous training as well as existing expertise are protected. At the same time, a monotonic extension is offered that does not require extra effort, but is likely to improve communication through models and teaching. A corresponding narrative could elaborate on the known problems people have with understanding conceptual models and on the need for lowering the level of abstraction without giving up on model quality. Figure 7 illustrates how a UML class diagram editor can be extended to integrate the representation of executable objects on M0 without mentioning levels above M1. This is especially helpful for students in an introductory course on object-oriented modeling. Not only that they see the relationship between classes and objects, they can also change the structure and behavior of objects by modifying the corresponding model. The screenshot was taken from the XModeler^{ML}. This is not possible with traditional tools, because that would require implementation languages, which, at least, feature classes on M2, cf. Sect. 4.

5.2 Relieving the pain: focus on frustration with the traditional approach

The primary target group of this strategy are language developers and ambitious and reflective modelers who are unsatisfied with, if not frustrated by limitations of traditional (meta) modeling languages that compromise the quality of languages and models. Professionals in this group have likely experienced serious problems. A common problem, which modelers who design information systems are familiar with, is caused by the lack of abstraction of both traditional modeling languages and implementation languages. A typical example is product types. Often, there is need to distinguish between different product types. However, especially if it is required to distinguish particular product instances, there is a lack of a further (meta) classification. In addition, the lack of further meta levels is a serious limitation of programming languages, too.

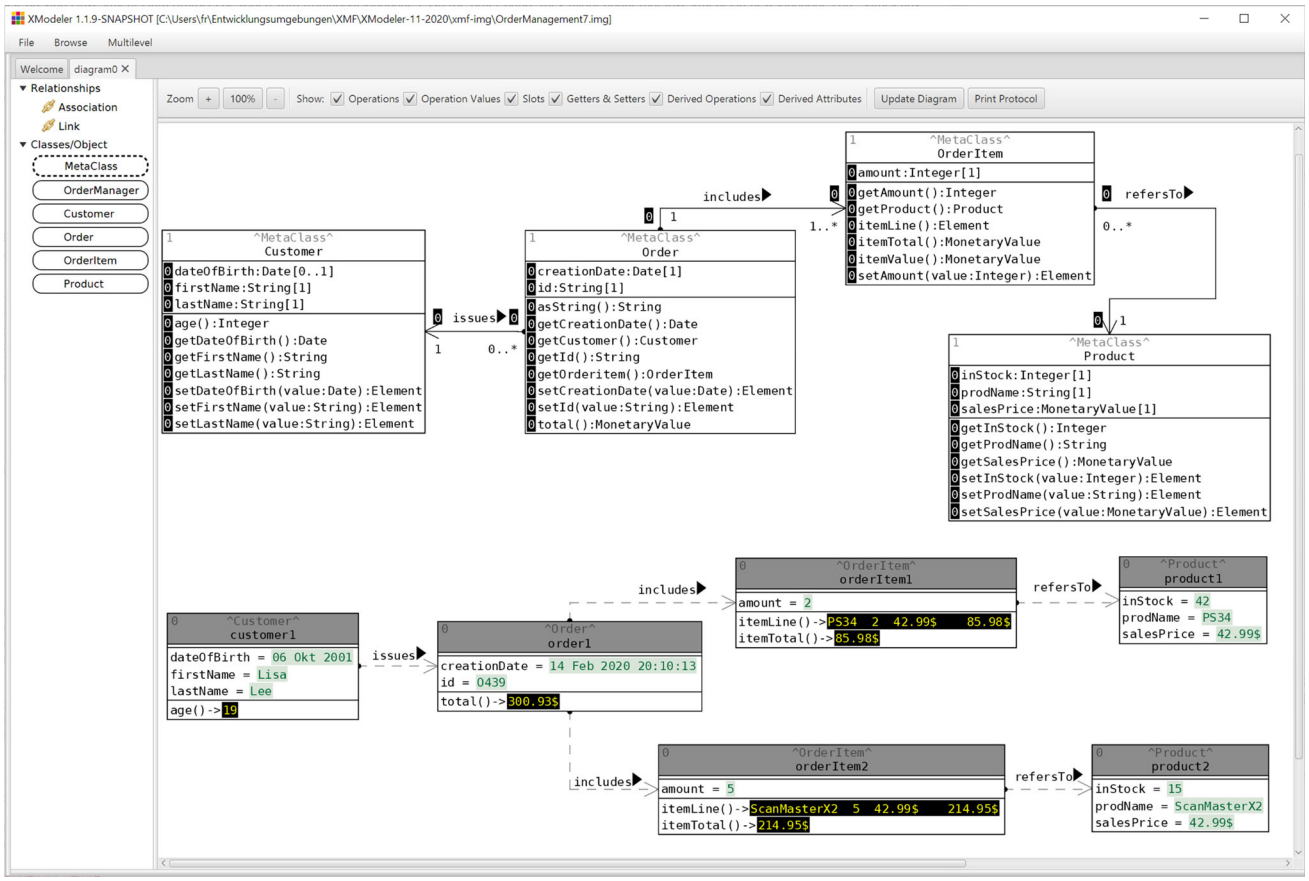

Fig. 7 Extending traditional object-oriented modeling: integration of objects and classes

This aspect of the prototypical strategy suggests developing a collection of known problems together with an illustration of how to overcome the identified limitations with multi-level modeling and, in the ideal case, with additional multi-level programming languages. Such an approach could build on several examples that exist to motivate the use of multi-level modeling, e.g. [18,44,60,67,90,98], and the patterns suggested by [31].

Language designers in particular have likely experienced various serious problems. At first, traditional language architectures are characterized by a clear dichotomy between modeling language and model. That does not only create the problem how to decide whether a concept should be part of a DSML or rather be specified with it, it also produces a principal design conflict, cf. Sect. 3. Second, DSMLs are usually specified from scratch using generic metalanguages. As argued in Sect. 3, it would often be more efficient to use more general DSMLs for the specification of more specific ones, which would result in a hierarchy of language layers. Third, the specification of DSMLs is frequently compromised by a lack of expressiveness, such as there is knowledge about a domain that applies to all instances of models, but it cannot be expressed in the DSML.

Figure 8 shows how multi-level modeling is suited to directly address the above problems with the traditional paradigm and, thus, offer language designers immediate benefits. If, for example, a language designer wants to specify a language for modeling IT infrastructures, the traditional paradigm would force her to start from scratch with concepts such as “class” and “attribute.” In the case of a multi-level language hierarchy, she could start with the lowest language level that still satisfies the specific requirements she has to account for, which would serve as the (meta) language to specify a more specific language. That corresponds directly to the development of specific technical terminologies in natural language. Usually, one would start with a more general language as they are, e.g., used in text books to create more specific terms, which could be further refined to account for more specific requirements. The example also illustrates their reuse and the refinement of notation elements. Since a language may comprise more than one level, there is not clear distinction between different languages.

Protection of investments and transition costs must be accounted for, too. Therefore, it seems reasonable to enable a step-wise transition to multi-level modeling that starts with modest extensions of existing conceptual models and

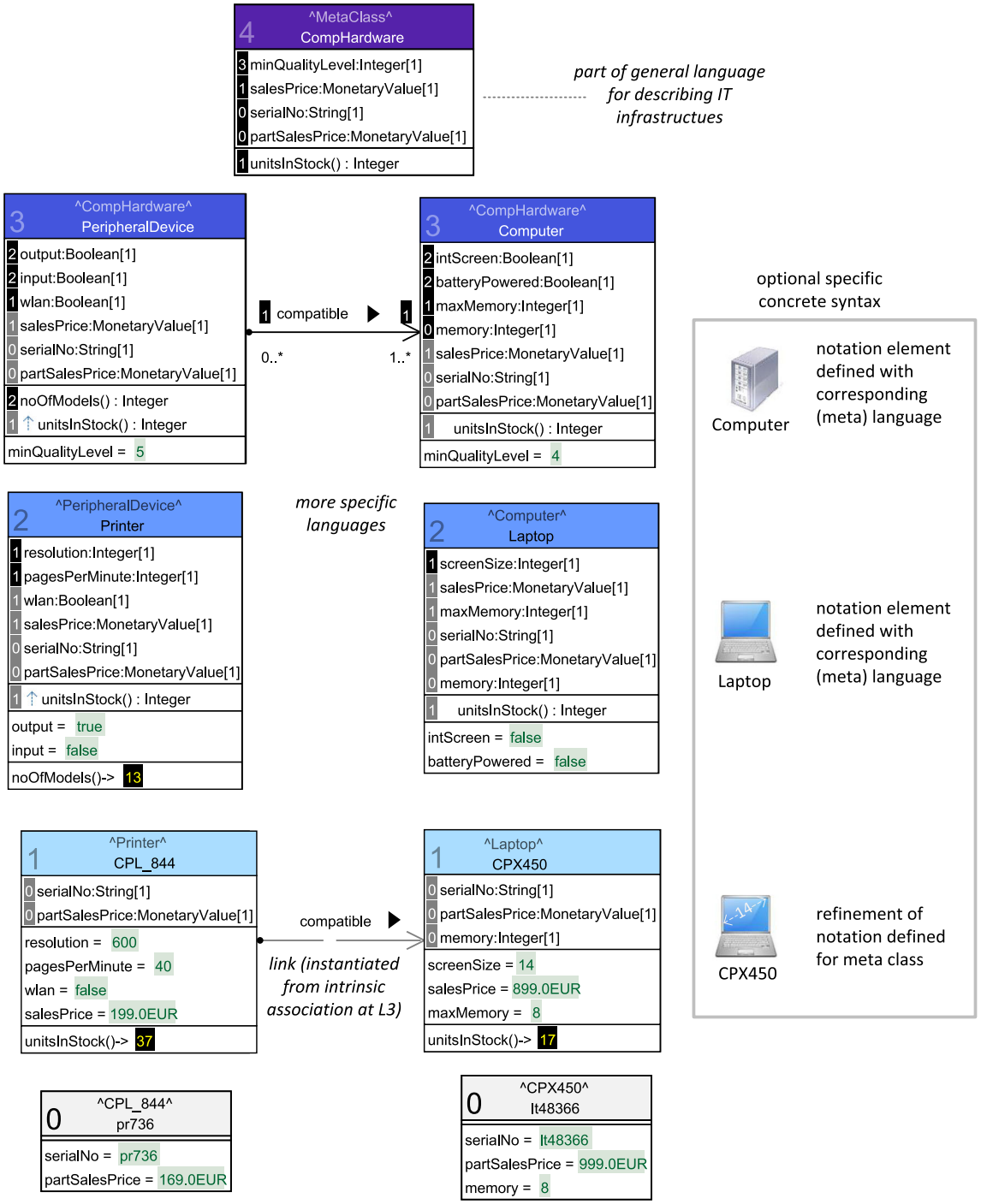

Fig. 8 No strict dichotomy of model and language(s)

language specifications. Figure 9 illustrates how this can be achieved in a multi-level modeling tool such as the XModeler^{ML}. The introduction of additional (meta) classes is not monotonic, but at first the changes that need to be made to an existing model could be kept small. The screenshots in Fig. 9 also illustrate the benefits of a common representation of models and programs. First, a model can be directly executed without the need to generate code. Second, depending on users' preferences, a model can be edited either with the diagram editor or with a model browser that includes an editor for code and constraints. Hence, the distinction between modeling and coding gets blurred, which is suited to narrow the gap between developers and users. Finally, the resulting software is integrated not only with its conceptual model, but also with the language(s), the model was designed with. Thereby, users are empowered not only to open the blackbox

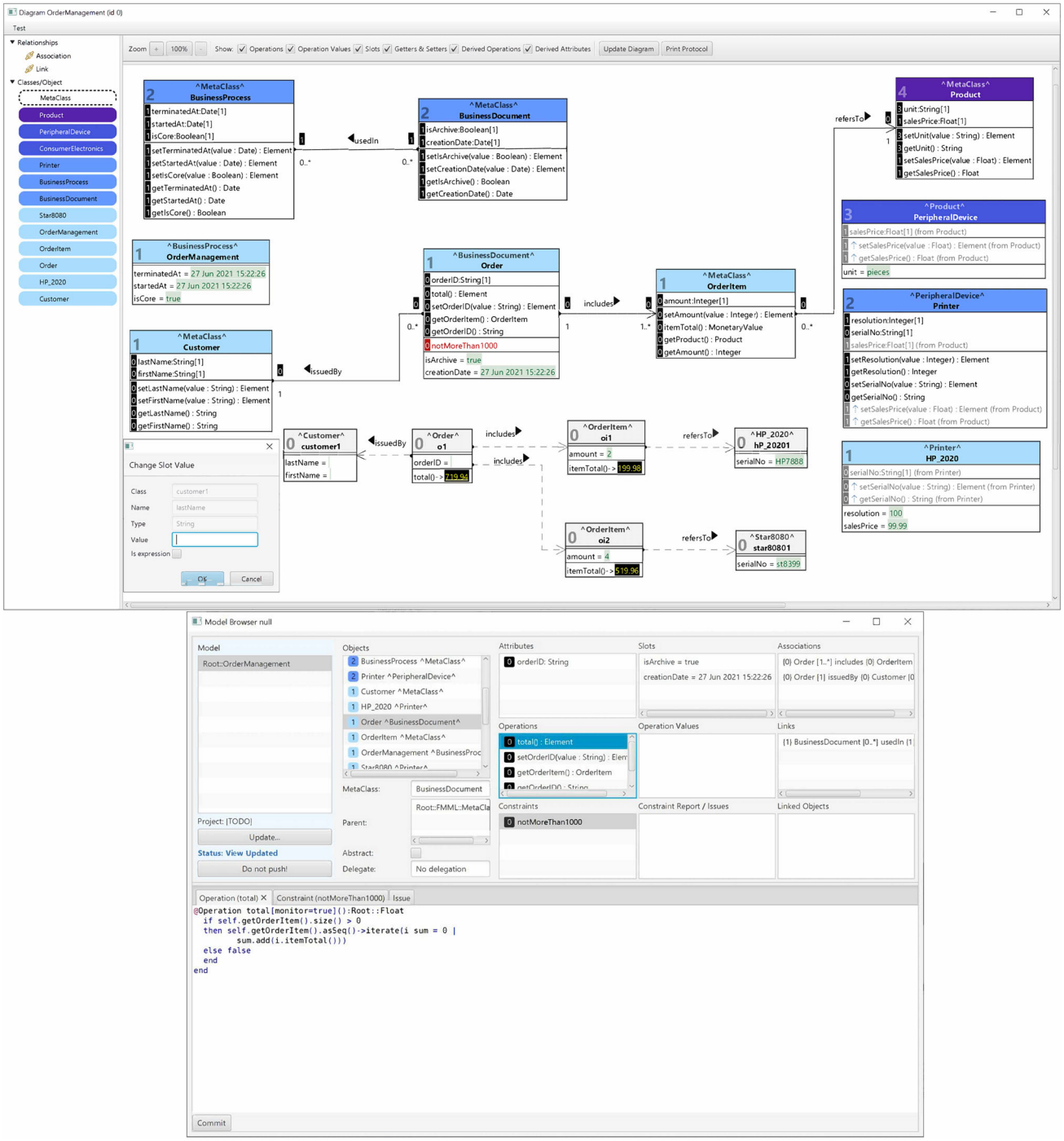

Fig. 9 Enriching developer and user experience by integrating modeling and coding

as which they usually experience software, but also to change it to a certain extent at run time.

While this target group is probably receptive to reasonable arguments, additional narratives should be useful, too. They could address common professional practices, typical use cases, but also account for conceptual hierarchies in natural language.

5.3 Invigorating a great idea: reference models reloaded

There is hardly any other idea in business informatics that received as much attention and popularity as reference models [15,16,36,50]. On the one hand, it was motivated by sobering empirical insights. Despite the considerable ben-

efits offered by conceptual models in general, and enterprise models in particular, many companies did not intend to work with conceptual models. This was due to the substantial effort required to develop and maintain large models and the risk of failure. On the other hand, reuse has been seen as a silver bullet of software development for long, since it promises to achieve two goals simultaneously that are usually in conflict: high quality through thorough, professional development, and relatively low acquisition costs through economies of scale. Reference models were almost unanimously welcomed by the scientific community and they also met with a positive response in many companies. Research focused on the development of reference models and on approaches to customize them (e.g., [38] presents a catalog of 38 reference models). Despite obvious benefits and broad support in academia, reference models did not become the game changers they were supposed to be. An initiative that adapted the idea of open source software to reference models [54] was not successful either. The limited adaptability of reference models turned out to be a notable obstacle. Modifications, such as specialization, are restricted to the same level of classification. If a reference model does not fit the classification required in a specific case, it is not possible—apart from the extensions of existing classifications through specialization—to adapt the model through the introduction of new classes. That would require at least one more level of classification. This is what DSMLs aim at. However, they suffer from the fact that they still require the manual creation of models. Therefore, supplementing reference models with DSMLs is suited to improve the adaptability of reference models. However, in traditional language architectures that would still create adaptability and integrity issues (see Sect. 3).

If a reference model does not fit specific needs, modifying it with a corresponding DSML should be safer and more convenient than using a GPML. However, a DSML is likely not specific enough to enable efficient modifications and to prevent inappropriate changes. This would, for example, be the case, when a reference model of organizational structures for a specific domain is supplemented with an organization modeling language. Multi-level reference models would allow integrating various DSMLs and specific reference models. If a reference model on the lowest level needs to be adapted and specialization is not sufficient, a more specific DSML could be used to guide the adaptation and to promote its consistency. A demonstration of this idea applied to the domain of smart grids is presented in [32]. The example shown in Fig. 8 illustrates that a multi-level model can integrate DSMLs and reference models, thus combining their specific advantages.

The main target group of this prototypical strategy would be the modeling community in business informatics. Demonstrating the potential of multi-level modeling to overcome the obstacles that prevented the success of reference model could be supplemented by an appealing narrative: "After all, it was a great idea, only the required language architecture was not available back then."

5.4 Focus on user experience and empowerment: prospects of multi-Level system architectures

So far, multi-level modeling has been mainly regarded as a powerful tool for advanced modelers. Even with respect to that group, there was a concern that the complexity of multi-level models would overwhelm users. It may therefore seem absurd to offer multi-level modeling to users who do not have a specific qualification. This is certainly a valid point. However, it does not have to prevent the use of multi-level models by non-expert users. In general, the reduction of complexity implies the increase of complexity at first. Usually, the construction of a tool that allows reducing the complexity of those tasks it is supposed to support will involve a level of complexity that is clearly beyond the complexity of the targeted tasks. In other words, while the foundational construction of multi-level language environments involves additional complexity, this complexity can be hidden more or less depending on the needs and skills of prospective users, who, at the same, time benefit from the reduction of complexity through additional abstraction.

Many scenarios are conceivable to demonstrate the utility of multi-level systems for users without specific modeling competence. One example scenario would be the use of a multi-level model, or in other words of multi-level DSMLs, by domain experts to create and extend models of the domains they are familiar with. For that purpose, they do not have to work with a diagram. Figure 10 illustrates how a multi-level model can be accessed and modified through a common GUI. From a technical point of view, this would enable domain experts to modify the (multi-level) schema of an information system, which would either require the generation of an implementation level representation or the common representation of models and programs, cf. Sect. 4. Users are not bothered with the peculiarities of language semantics or notation. Instead, they work with the concepts they are familiar with. The tool guides them with structuring the domain appropriately.

A further scenario concerns a multi-level architecture of enterprise software systems such as ERP systems. The use of these systems is restricted to objects on MO. The schema level (M1) is usually hidden from users. While hiding higher levels of classification from users might contribute to a reduction of complexity, it has two notable shortcomings. First, these systems remain to a large extent black boxes to the users. The conception users can develop of a system is restricted to the user interface they experience, and is typically confined to those use cases they are familiar with. Second, as a consequence of the first, most users do not know

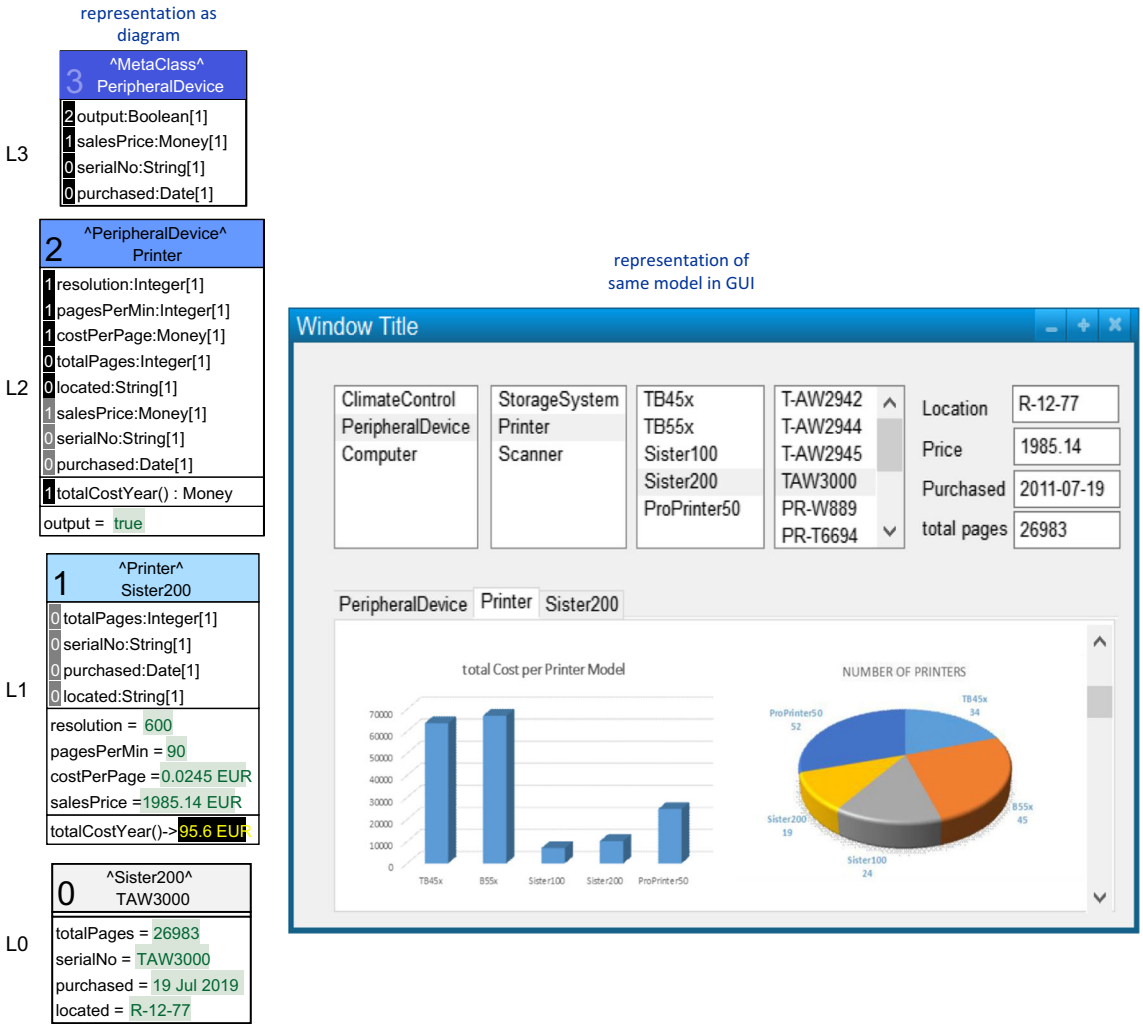

Fig. 10 Implicit multi-level modeling and corresponding diagram

whether and how a system can be modified. With respect to the ongoing penetration of organizations with software, both aspects have the potential to increase employees' alienation not only from a technology they do not understand, but from the surroundings they work in. In addition, it hinders the efficient modification of systems by those who know the domain and the requirements. A multi-level system architecture would enable navigating the conceptual foundations of a model through introspection. An object can be asked for the properties of its class, which in turn can be asked for its (meta) class. To take advantage of introspection for non-expert users, it would be required to represent the various levels of a system's conceptual foundation in an accessible way, e.g., by diagrams created with DSMLs or browsers like the one shown in Fig. 10.

This idea corresponds to the conception of "self-referential enterprise systems" (SRES), which has been proposed some time ago already [42]. It is based on the integration of an enterprise software system with an enterprise model that provides a conceptualization of both, the software *and* the organizational action system, the software operates in (see, e.g., [52]). The benefits of SRES were demonstrated by various use cases, cf. [42]. However, the proposed software architectures were clearly limited, because they were based on a MOF-like language hierarchy. Multi-level system architectures would enable the common representation of models and programs at run time. In addition, they would allow for navigating multiple levels of classification and for modifying models—provided a user has the knowledge of the affected domain concepts and the required authorization.

This strategy could be supported by a narrative on the fact that software is more and more determining the way we work and make decisions. Therefore, reducing software to black boxes is a serious threat to self-determination and the ideas of enlightenment in general. Opening the black boxes through accessible representations would contribute to empowering users by fostering a deeper understanding of software and its adaptability.

6 Conclusions and future work

This paper presents a rationale for multi-level modeling that demonstrates its strengths and prospects and that is suited to motivate an intensification of current research efforts. To that end, we show that multi-level modeling represents an outstanding, if not the most important contribution to conceptual modeling in the recent past. Among other things, the additional abstraction it allows for enables relaxing essential design conflicts. It is also suited to combine specific benefits of conceptual models with those of DSMLs. Different from traditional DSMLs, the design of multi-level DSMLs does not suffer from the need to artificially distinguish between language specification and language application.

Moreover, our recourse to findings in cognitive linguistics and cognitive psychology invalidates a reservation sometimes expressed against multi-level modeling, namely that the distinction of multiple levels of classification does not fit human cognition. Instead, there are indications that the alienation sometimes triggered by multi-level models is not due to their contrast to natural language, but rather to their contrast to traditional approaches. While multi-level modeling can indeed be seen as a new paradigm, this does not mean that it can blossom only after the prevailing paradigm has been defeated. Different from Kuhn's conception of scientific progress through a war of paradigms, the multi-level community does not have an "independent existence" [71, p. 117] that separates it from other communities. Furthermore, multi-level modeling can be seen as an almost monotonic extension of traditional object-oriented modeling. Therefore, there is no need to fight the current paradigm.

The prospects of multi-level modeling are contrasted by challenges that are not easy to overcome. With respect to protection of investment, one could demand for the standardization of multi-level modeling languages. However, we believe that it is too early for freezing the current state. Instead, there is need for competition that is based on the spirit of common convictions to address remaining research questions (see below). For that purpose, the dissemination of multi-level modeling in academia and practice is of pivotal relevance. We suggest four prototypical strategies to build incentives for using multi-level modeling. We also show that supplementing multi-level modeling languages with multi-level programming languages is a promising approach in that respect.

To promote the dissemination of multi-level modeling, it is important that it is accounted for in curricula of university programs. That recommends at first the development of a consistent and satisfactory common terminology, which would at best be organized as a community project. In addition, there is need for the development of specific analysis and design methods. Furthermore, the collective design and implementation of a prototypical multi-level application system, e.g., a small ERP system, could serve as both, an open laboratory for testing and further developing multi-level concepts and tools, and as a showcase to demonstrate specific benefits enabled by multi-level architectures. To that end, more research is needed to assess and, at best, measure the effect of additional levels of (intrinsic) classification with a high degree of validity and reliability.

To take advantage of the additional abstraction enabled by multi-level modeling, elaborate domain theories are mandatory. The development of domain theories falls in the realm of other disciplines. At the same time, structuring a domain theory with respect to the abstraction concepts provided by multi-level modeling requires specific modeling expertise. Therefore, cross-disciplinary collaboration would be especially important. The digital transformation is suited to make such a collaboration particularly exciting for researchers from all participating fields: There is not only need for theories that describe or explain domains as they are, but also as they might be in the future in order to investigate possible paths of change.

While core concepts of multi-level modeling have reached a rather mature state, there are still appealing opportunities for future research. The semantics of contingent level classes, or more general, of models that include contingencies, still requires further clarification. For those approaches that are based on logic, there might be the option to include aspects of modal logic. Approaches that use object-oriented concepts may require splitting contingent models, e.g., by defining separate name spaces that are used to manage particular, consistent parts of contingent models. Even for multi-level models, which have been developed with great care, it may turn out that aspects that were assumed to be invariant have to be changed. Due to multiple, cross-level dependencies, refactoring multi-level models creates considerable challenges, which recommends further research into strategies that support efficient and consistent management and change of models. For an approach to address this challenge with a pattern catalogue, see [29].

Since the current friction between multi-level models and prevalent programming languages is suited to compromise the advantages of multi-level models as a foundation of software systems, more research on multi-level (meta) programming languages is required. That includes the analysis of static typing vs. dynamic typing and the integration with modeling languages. With respect to the prospects of multi-level system architectures, research on common representations of models and programs at run time is especially relevant, because it opens the perspective on software that empowers its users. Related to that, research on architectures that enable substantial changes—by taking advantage of the abstraction provided by higher levels of an architecture—is especially promising, because it would address the need to

cope with fundamental changes as they may be required by the digital transformation.

Machine learning builds the foundation for a different stream of research on multi-level modeling. Even though we are reluctant to believe in Domingos's prediction that machine learning will replace knowledge engineers in the near future [33, p. 36], the use of inductive approaches, provided appropriate data are available, might help with developing candidates for higher-level classes.

Finally, more research is required to address one of the biggest challenges, which is at the same time of great importance in promoting the usefulness of multi-level modeling. Previous research was mainly focused on static abstractions. The development of multi-level dynamic abstractions, that is, multi-level process models could contribute to clearly raise reuse and adaptability of process types. A Dagstuhl seminar that was dedicated to this subject [2] and a specific multi-modeling contest [1] confirmed the relevance of multi-level dynamic abstractions as well as the great difficulties their construction faces—which translates to a promising research opportunity.

Acknowledgements I am grateful to Monika Kaczmarek-Heß and Sybren de Kinderen for numerous valuable comments on an earlier version of the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Almeida, J.P.A., Rutle, A., Wimmer, M., Kühne, T.: The multi process challenge (2019)
2. Almeida, J.P.A., Frank, U., Kühne, T.: Multi-level modelling (Dagstuhl Seminar 17492). Dagstuhl Rep **7**(12), 18–49 (2018)
3. Aßmann, U., Götz, S., Jézéquel, Morin, B., Trapp, M.: A reference archtitecture and roadmap for models@run.time systems. In: Bencomo, N., France, R., Cheng, B.H.C., Aßmann, U. (eds.) Models@run.time, State-of-the-art Survey, pp. 1–18. Springer, Cham (2014)
4. Atkinson, C., Gerbig, R., Kühne, T.: Comparing multi-level modeling approaches. In: Proceedings of the Workshop on Multi-Level Modelling (MULTI 2014), Valencia, Spain, vol. 1286, pp. 53–61. Aachen (2014). https://madoc.bib.uni-mannheim.de/43516/
5. Atkinson, C., Gerbig, R.: Flexible deep modeling with Melanee. In: Reimer, U., Betz, S. (eds.) Modellierung 2016, 2.–4. März 2016, Karlsruhe—Workshop Proceedings, Modellierung 2016, vol. 255, pp. 117–122. Gesellschaft für Informatik, Bonn (2016)
6. Atkinson, C., Gerbig, R.: Melanie: multi-level modeling and ontology engineering. In: Proceedings of the 2nd International Master Class on Model-Driven Engineering Modeling Wizards, pp. 1–2. ACM, New York (2012)
7. Atkinson, C., Kühne, T.: On evaluating multi-level modeling. In: Burgueño, L., et al. (eds.) Proceedings of the 4th International Workshop on Multi-Level Modeling (MULTI), pp. 274–277. CEUR-WS.org (2017)
8. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: Gorgolla, M., Kobryn, C. (eds.) The Unified Modeling Language. Modeling Languages, Concepts, and Tools. Lecture Notes in Computer Science, pp. 19–33. Springer, Berlin (2001)
9. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Softw. Syst. Model. **7**(3), 345–359 (2008)
10. Atkinson, C., Kühne, T., Lara, J.D.: Editorial to the theme issue on multi-level modeling. Softw. Syst. Model. **17**(1), 163–165 (2018)
11. Balaban, M., Khitron, I., Kifer, M., Maraee, A.: Multilevel modeling: what's in a level? A position paper. In: Hebig, R., Berger, T. (eds.) Proceedings of MODELS 2018 Workshops, CEUR Workshop Proceedings, vol. 2245, pp. 693–697. CEUR-WS.org (2018)
12. Bateson, G.: Steps to an Ecology of Mind, 1st edn. Ballantine, New York (1978)
13. Batra, D., Davis, J.G.: Conceptual data modelling in database design: similarities and differences between expert and novice designers. Int. J. Man Mach. Stud. **37**(1), 83–101 (1992)
14. Beck, K., Beedle, Mike, et al.: Manifesto for Agile Software Development. Technical reports (2001). https://agilemanifesto.org/
15. Becker, J., Algermissen, L., Niehaves, B., Delfmann, P.: Business process reference models for reorganizing public administrations—a case study. In: Andersen, K., Grönlund, A., Traunmüller, R., Wimmer, M. (eds.) Electronic Government—Workshop and Poster Proceedings of the Fourth International EGOV Conference 2005, August 22–26, 2005, Copenhagen, Denmark, Schriftenreihe Informatik, vol. 13, pp. 134–142. Universitätsverlag Rudolf Trauner, Linz, Austria (2005)
16. Becker, J., Delfmann, P. (eds.): Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models. Physica Verlag, Heidelberg (2007)
17. Bencomo, N., France, R., Cheng, B.H.C., Aßmann, U. (eds.): Models@run.time: foundations, applications, and roadmaps—Dagstuhl Seminar 11481 on Models@Run.Time, November 27 to December 2, 2011. In: State-of-the-art Survey, vol. 8378. Springer, Cham (2014)
18. Benner, B.: A multilevel approach for model-based user interface development. In: Clark, T., Frank, U., Wimmer, M. (eds.) Proceedings of the 4th International Workshop on Multi-Level Modelling (MULTI) 2017 (2017)
19. Bock, A.C.: The power/generality trade-off in decision and problem modeling: theoretical background and multi-level modeling as a resolution. In: Gulden, J., Reinhartz-Berger, I., Schmidt, R., Guerreiro, S., Guédria, W., Bera, P. (eds) Enterprise, Business-Process and Information Systems Modeling, Lecture Notes in Business Information Processing, vol. 318, pp. 213–228. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-91704-7_14
20. Booch, G.: Object-Oriented Analysis and Design with Applications, 2nd edn. Benjamin Cummings, Redwood City (1994)
21. Brunelière, H., Cabot, J., Dupé, G., Madiot, F.: MoDisco: a model driven reverse engineering framework. Inf. Softw. Technol. **56**(8), 1012–1032 (2014)

22. Bruner, J.: The narrative construction of reality. Crit. Inq. **18**(1), 1–21 (1991)
23. Bunge, M.: Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World. Reidel, Dordrecht (1977)
24. Cheng, B.H.C., Eder, K.I., Gogolla, M., Grunske, Lars, et al.: Using models at runtime to address assurance for self-adaptive systems. In: Bencomo, N., France, R., Cheng, B.H.C., Aßmann, U. (eds.) Models@run.time, State-of-the-art Survey, pp. 101–136. Springer, Cham (2014)
25. Clark, T., Sammut, P., Willans, J.: Applied Metamodelling: A Foundation for Language Driven Development. Ceteva, Sheffield (2008). http://eprints.mdx.ac.uk/6060/. First edn. published 2004
26. Clark, T., Sammut, P., Willans, J.: Superlanguages: developing languages and applications with XMF. Ceteva (2008)
27. de Kinderen, S., Kaczmarek-Heß, M.: On model-based analysis of organizational structures: an assessment of current modeling approaches and application of multi-level modeling in support of design and analysis of organizational structures. Softw. Syst. Model. **19**(2), 313–343 (2020)
28. de Lara, J., Guerra, E.: Deep meta-modelling with metadepth. In: Vitek, J. (ed.) Objects, Models, Components, Patterns, pp. 1–20. Springer, Berlin (2010)
29. de Lara, J., Guerra, E.: Refactoring multi-level models. ACM Trans. Softw. Eng. Methodol. (TOSEM) **27**, 1–56 (2018)
30. de Lara, J., Guerra, E., Cobos, R., Moreno-Llorena, J.: Extending deep meta-modelling for practical model-driven engineering. Comput. J. **57**(1), 36–58 (2012). https://doi.org/10.1093/comjnl/bxs144
31. de Lara, J., Guerra, E., Cuadrado, J.S.: When and how to use multilevel modelling. ACM TOSEM **24**(2), 12:1-12:46 (2014)
32. de Kinderen, S., Kaczmarek-Heß, M.: Multi-level modeling as a language architecture for reference models: on the example of the smart grid domain. In: Becker, J., Novikov, D. (eds) Proceedings of the 21st IEEE Conference on Business Informatics, CBI 2019, Volume 1—Research Papers, Moscow, pp. 174–183. IEEE (2019)
33. Domingos, P.: The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. Penguin Books Ltd, London (2017)
34. Evans, J., Over, D.E., Manktelow, K.I.: Reasoning, decision making and rationality. Cognition **49**(1), 165–187 (1993)
35. Fauconnier, G., Turner, M.: The way we think: conceptual blending and the mind's hidden complexities. Basic Books (2002)
36. Fettke, P., Loos, P. (eds.): Reference modeling for business systems analysis. Idea Group, Hershey (2007)
37. Fettke, P., Loos, P.: Ontological evaluation of reference models using the Bunge-Wand-Weber-model. In: Proceedings of the Ninth Americas Conference on Information Systems (AMCIS 2003), pp. 2944–2955. AIS (2003)
38. Fettke, P., Loos, P.: Referenzmodellierungsforschung. Wirtschaftsinformatik **46**(5), 331–340 (2004)
39. Florian, H., Jendrik, J., Mirko, S., Christian, W.: Construct to reconstruct—reverse engineering java code with JaMoPP. In: Proceedings of the International Workshop on Reverse Engineering Models from Software Artifacts, R.E.M. 2008, Satellite event of the 16th Working Conference on Reverse Engineering (WCRE), October 15, 2009, Lille, France (2009)
40. Fonseca, C.M., Almeida, J.P.A., Guizzardi, G., de Carvalho, V.A.: Multi-level conceptual modeling: from a formal theory to a well-founded language. In: Trujillo, J., Davis, K.C.E.A. (eds) Proceedings of 37th International Conference on Conceptual Modeling (ER 2018), Lecture Notes in Computer Science, vol. 11157, pp. 409–423. Springer (2018)
41. France, R.B., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: Briand, L.C., Wolf, A.L. (eds.) Workshop on the Future of Software Engineering (FOSE, vol. '07), pp. 37–54. IEEE CS Press (2007)
42. Frank, U., Strecker, S.: Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems: Requirements, Conceptual Foundation and Design Options. Technical Reports 31, Institut für Informatik und Wirtschaftsinformatik (ICB), Universität Duisburg-Essen, Essen (2009)
43. Frank, U., Töpel, D.: Contingent level classes: motivation, conceptualization, modeling guidelines, and implications for model management. In: Guerra, E., Iovino, L. (eds) Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 622–631. New York, NY, USA (2020)
44. Frank, U.: Designing models and systems to support IT management: a case for multilevel modeling. In: Proceedings of MULTI 2016, pp. 3–24. CEUR-WS.org (2016)
45. Frank, U.: Domain-specific modeling languages—requirements analysis and design guidelines. In: Reinhartz-Berger, I., Sturm, A., Clark, T., Wand, Y., Cohen, S., Bettin, J. (eds.) Domain Engineering: Product Lines, Conceptual Models, and Languages, pp. 133–157. Springer (2013)
46. Frank, U.: Evaluating Modelling Languages: Relevant Issues, Epistemological Challenges and a Preliminary Research Framework. Technical Reports 15. Institut für Wirtschaftsinformatik, Universität Koblenz-Landau, Koblenz (1998)
47. Frank, U.: MEMO Organisational Modelling Language: Requirements and Core Diagram Types. Technical Reports 47, ICB University of Duisburg-Essen, Campus Essen (2011)
48. Frank, U.: Prolegomena of a multi-level modeling method. illustrated with the FMML^{x} and the XModeler^{ML}. In: Proceedings of the 24rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (2021)
49. Frank, U.: The Flexible Modelling and Execution Language (FMML^{x})—Version 2.0: Analysis of Requirements and Technical Terminology. Technical Reports 66, University Duisburg-Essen (2018)
50. Frank, U.: A conceptual foundation for versatile E-Commerce platforms. J. Electron. Commer. Res. **2**(2), 48–57 (2002)
51. Frank, U.: Evaluation of reference models. In: Fettke, P., Loos, P. (eds.) Reference Modeling for Business Systems Analysis, pp. 118–140. Idea Group, Hershey (2007)
52. Frank, U.: Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Softw. Syst. Model. **13**(3), 941–962 (2014)
53. Frank, U.: Multilevel modeling: toward a new paradigm of conceptual modeling and information systems design. BISE **6**(6), 319–337 (2014)
54. Frank, U., Strecker, S.: Open reference models: community-driven collaboration to promote development and dissemination of reference models. Enterp. Model. Inf. Syst. Archit. **2**(2), 32–41 (2007)
55. Garson, J.: Modal logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University (2021). https://plato.stanford.edu/archives/sum2021/entries/logic-modal/
56. Goldstein, R.C., Storey, V.C.: Materialization. IEEE Trans. Knowl. Data Eng. **6**(5), 835–842 (1994)
57. Green, V.E., Evans, V.: Cognitive Linguistics: An Introduction. Edinburgh University Press, Edinburgh (2007)
58. Grossmann, R.: The Categorical Structure of the World. Indiana University Press, Bloomington (1983)
59. Guizzardi, G., Herre, H., Wagner, G.: On the general ontological foundations of conceptual modeling. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) Conceptual modeling-ER 2002. Lecture Notes in Computer Science, vol. 2503, pp. 65–78. Springer, Berlin (2002)

60. Igamberdiev, M., Grossmann, G., Stumptner, M.: A Feature-based categorization of multi-level modeling approaches and tools. In: Atkinson, C., Grossmann, G., Clark, T. (eds) Proceedings of the 3rd International Workshop on Multi-Level Modelling (MULTI 2016), Saint-Malo, France, CEUR Workshop Proceedings, vol. 1722, pp. 45–55. CEUR-WS.org (2016)
61. Jácome-Guerrero, S.P., de Lara, J.: TOTEM: reconciling multi-level modelling with standard two-level modelling. Comput. Stand. Interfaces **69**, 103390 (2020)
62. Jarke, M., Jeusfeld, M., Nissen, H., Quix, C., Staudt, M.: Metamodelling with datalog and classes: conceptbase at the age of 21. In: Norrie, M., Grossniklaus, M. (eds.) Object Databases. Lecture Notes in Computer Science, vol. 5936, pp. 95–112. Springer, Berlin (2010)
63. Jarke, M., Eherer, S., Gallersdörfer, R., Jeusfeld, M., Staudt, M.: ConceptBase: a deductive object base for meta data management. J. Intell. Inf. Syst. **4**(2), 167–192 (1995)
64. Jeusfeld, M.A., Neumayr, B.: DeepTelos: Multi-level modeling with most general instances. In: Comyn-Wattiau, I., Tanaka, K., Song, I.Y., Yamamoto, S., Saeki, M. (eds) Proceedings of the 35th International Conference on Conceptual Modeling (ER 2016), pp. 198–211. Springer, Cham (2016)
65. Jeusfeld, M.: DeepTelos for ConceptBase: a contribution to the multi process challenge. In: Proceedings of MULTI 2019. CEUR-WS.org (2019)
66. Kaczmarek-Heß, M., Nolte, M., Fritsch, A., Betz, S.: Practical experiences with multi-level modeling using FMML^{x}: a hierarchy of domain-specific modeling languages in support of life-cycle assessment. In: Clark, T., Neumayr, B., Rutle, A. (eds.) Proceedings of the 5th International Workshop on Multi-Level Modelling (MULTI) 2018 (2018)
67. Kaczmarek-Heß, M., de Kinderen, S.: A multilevel model of IT platforms for the needs of enterprise IT landscape analyses. Bus. Inf. Syst. Eng. **59**(5), 315–329 (2017)
68. Kahneman, D., Slovic, P., Tversky, A. (eds.): Judgment Under Uncertainty: Heuristics and Biases. Cambridge University Press, Cambridge (1982)
69. Kärnä, J., Tolvanen, J.P., Kelly, S.: Evaluating the use of domain-specific modeling in practice. In: The 9th OOPSLA Workshop on Domain-Specific Modeling (2009)
70. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-Interscience and IEEE Computer Society, Hoboken (2008)
71. Kuhn, T.S.: The Structure of Scientific Revolutions. University of Chicago Press, Chicago (1964)
72. Kühne, T.: A story of levels. In: Hebig, R., Berger, T. (eds) Proceedings of MODELS 2018 Workshops, CEUR Workshop Proceedings, vol. 2245, pp. 673–682. CEUR-WS.org (2018)
73. Kühne, T., Schreiber, D.: Can programming be liberated from the two-level style: multi-level programming with deepjava. ACM SIGPLAN Notices **42**(10), 229–244 (2007)
74. Lakoff, G.: Cognitive versus generative linguistics: How commitments influence results. Lang. Commun. **1**(1) (1990)
75. Lakoff, G.: Women, Fire and Dangerous Things: What Categories Reveal About the Mind. University of Chicago Press, Chicago (1987)
76. Lakoff, G.: Women, Fire and Dangerous Things: What Categories Reveal About the Mind, 1st edn. University of Chicago Press, Chicago (1990)
77. Lakoff, G., Johnson, M.: Metaphors we live by. University of Chicago Press, Chicago (2008)
78. Langacker, R.W.: Foundations of Cognitive Grammar: Theoretical Prerequisites, vol. 1. Stanford University Press, Palo Alto (1987)
79. Lange, A., Atkinson, C.: Multi-level modeling with MELANEE. In: Proceedings of MODELS 2018 Workshops, vol. 2245. CEUR-WS.org (2018)
80. Lethbridge, T.C., Abdelzad, V., Husseini Orabi, M., Husseini Orabi, A., Adesina, O.: Merging modeling and programming using Umple. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications, pp. 187–197. Springer International Publishing, Cham (2016)
81. Macías, F., Rutle, A., Stolz, V.: MultEcore: Combining the Best of Fixed-Level and Multilevel Metamodelling. In: C. Atkinson, G. Grossmann, T. Clark (eds.) Proceedings of the 3rd International Workshop on Multi-Level Modelling, Saint-Malo,France, October 4, 2016, *CEUR Workshop Proceedings*, vol. 1722, pp. 66–75. CEUR-WS.org (2016)
82. Macías, F., Rutle, A., Stolz, V., Rodríguez-Echeverría, R., Wolter, U.: An approach to flexible multilevel modelling. Enterp. Model. Inf. Syst. Archit. **13**, 1–35 (2018)
83. Metzger, W., Stadler, M., Crabus, H. (eds.): Gestalt-Psychologie: Ausgewählte Werke aus den Jahren 1950–1982. Kramer, Frankfurt A. M. (1986)
84. Meyer, B.: Object-Oriented Software Construction, 2nd edn. Prentice Hall, Upper Saddle River (1997)
85. Neumayr, B., Grün, K., Schrefl, M.: Multi-level domain modeling with M-objects and M-relationships. In: Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling—Volume 96, APCCM '09, pp. 107–116. Australian Computer Society Inc., Darlinghurst, Australia (2009). http://dl.acm.org/citation.cfm?id=1862739.1862754
86. Neumayr, B., Schrefl, M.: Multi-level conceptual modeling and OWL. In: Heuser, C.A., Pernul, G. (eds.) Advances in Conceptual Modeling—Challenging Perspectives. Lecture Notes in Computer Science, pp. 189–199. Springer, Berlin (2009)
87. Neumayr, B., Schuetz, C.G., Horner, C., Schrefl, M.: DeepRuby: extending ruby with dual deep instantiation. In: Burgueño, L., Corley, J., Neumayr, B., et al. (eds.) Proceedings of MODELS 2017 Workshops, CEUR Workshop Proceedings, vol. 2019, pp. 252–260. CEUR-WS.org (2017)
88. Neumayr, B., Schuetz, C.G., Jeusfeld, M.A., Schrefl, M.: Dual deep modeling: multi-level modeling with dual potencies and its formalization in F-logic. Softw. Syst. Model. **17**(1), 233–268 (2018). https://doi.org/10.1007/s10270-016-0519-z
89. Newell, A.: Heuristic programming: Ill-structured problems. In: Aronofsky, J.S. (ed.) Progress in Operations Research. Relationship Between Operations Research and the Computer, pp. 361–414. Wiley, New York (1969)
90. Nolte, M., Kaczmarek-Heß, M., Fritsch, A., Betz, S.: A hierarchy of DSMLs in support of product life-cycle assessment. In: Ludwig, T., Pipek, V. (eds) Human Practice. Digital Ecologies. Our Future. Proceedings of the 14th Internationale Tagung Wirtschaftsinformatik (WI 2019), pp. 1433–1447. University of Siegen (2019)
91. Odell, J.J.: Power types. J. Object Orient. Program. **7**(2), 8–12 (1994)
92. OMG: Meta Object Facility (MOF) Core Specification: Version 2.0. Technical Report, Object Management Group (2006). http://www.omg.org/spec/MOF/2.0/
93. OMG: Model driven architecture (mda): Mda guide rev. 2.0. Tech. rep. (2014). https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf
94. Opdahl, A.L., Henderson-Sellers, B.: Ontological evaluation of the UML using the Bunge–Wand–Weber model. Softw. Syst. Model. **1**(1), 43–67 (2002). https://doi.org/10.1007/s10270-002-0003-9
95. Pirotte, A., Zimányi, E., Massart, D., Yakusheva, T.: Materialization: a powerful and ubiquitous abstraction pattern. In: Bocca,

J.B., Jarke, M., Zaniolo, C. (eds) Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pp. 630–641. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA (1994)
96. Popper, K.R.: The Logic of Scientific Discovery, 2, impr edn. Hutchinson, London (1960)
97. Rosch, E.: Principles of categorization. In: Rosch, E., Lloyd, B.B. (eds.) Cognition and Categorization, pp. 27–48. Erlbaum, Hillsdale (1978)
98. Rossini, A., de Lara, J., Guerra, E., Nikolov, N.: In: Taentzer, G., Bordeleau, F. (eds.) A comparison of two-level and multi-level modelling for cloud-based applications. In: Taentzer, G., Bordeleau, F. (eds) Modelling Foundations and Applications: Proceedings of the 11th European Conference, ECMFA 2015, Held as Part of STAF 2015, L'Aquila, Italy, pp. 18–32. Springer International Publishing, Cham (2015)
99. Schütte, R., Rotthowe, T.: The Guidelines of modeling—an approach to enhance the quality in information models. In: Ling, T.W., Ram, S., Li Lee, M. (eds.) Conceptual Modeling—ER '98, pp. 240–254. Springer Berlin (1998)
100. Selway, M., Stumptner, M., Mayer, W., Jordan, A., Grossmann, G., Schrefl, M.: A conceptual framework for large-scale ecosystem interoperability and industrial product lifecycles. Data Knowl. Eng. **109**, 85–111 (2017)
101. Siau, K., Loo, P.P.: Identifying difficulties in learning UML. Inf. Syst. Manag. **23**(3), 43–51 (2006)
102. Special Issue on Multi-Level Modeling: Enterprise Modelling and Information Systems Architectures **13** (2018)
103. Stahl, T., Völter, M.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester (2006)
104. Tuggy, D.: Schematicity. In: The Oxford Handbook of Cognitive Linguistics (2007)
105. Völter, M.: DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. dslbooks.org (2013)
106. Volz, B.W.: Werkzeugunterstützung für methodenneutrale Metamodellierung. Bayreuth (2011)
107. Wand, Y., Storey, V.C., Weber, R.: An ontological analysis of the relationship construct in conceptual modeling. ACM Trans. Database Syst. **24**(4), 494–528 (1999). https://doi.org/10.1145/331983.331989
108. Watzlawick, P., Weakland, J.H., Fisch, R.: Change: Principles of Problem Formation and Problem Resolution. Norton, New York (1974)
109. Yang, Y., Johnson-Laird, P.: Illusions in quantified reasoning: how to make the impossible seem possible, and vice versa. Mem. Cogn. **28**(3), 452–465 (2000)

Ulrich Frank holds the chair of Information Systems and Enterprise Modelling at the Institute of Computer Science and Business Information Systems at the University of Duisburg-Essen. His main research topic is enterprise modelling, i.e. the development and evaluation of modelling languages, methods and corresponding tools. In recent years, he focused especially on multi-level domain-specific modelling languages and corresponding tools. Further areas of research include method construction, (meta) programming languages, and advanced architectures of application systems. He is also interested in the philosophy of science and fundamental questions related to the subject of research in business information systems and computer science.