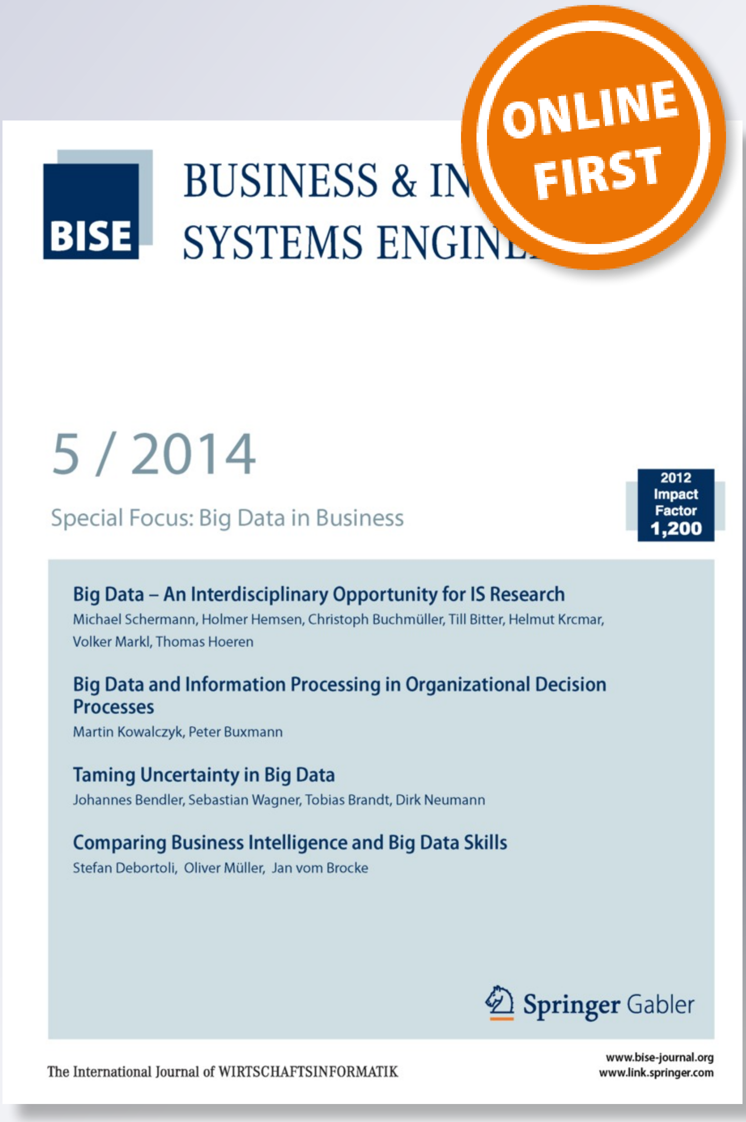
Multilevel Modeling

Ulrich Frank

Business & Information Systems Engineering
The International Journal of WIRTSCHAFTSINFORMATIK

e-ISSN 1867-0202

Bus Inf Syst Eng
DOI 10.1007/s12599-014-0350-4



BUSINESS & INFORMATION SYSTEMS ENGINEERING

ONLINE FIRST

5 / 2014

Special Focus: Big Data in Business

2012 Impact Factor 1,200

Big Data – An Interdisciplinary Opportunity for IS Research
Michael Schermann, Holmer Hemsen, Christoph Buchmüller, Till Bitter, Helmut Krcmar, Volker Markl, Thomas Hoeren

Big Data and Information Processing in Organizational Decision Processes
Martin Kowalczyk, Peter Buxmann

Taming Uncertainty in Big Data
Johannes Bendler, Sebastian Wagner, Tobias Brandt, Dirk Neumann

Comparing Business Intelligence and Big Data Skills
Stefan Debortoli, Oliver Müller, Jan vom Brocke

Springer Gabler

The International Journal of WIRTSCHAFTSINFORMATIK

www.bise-journal.org
www.link.springer.com

Multilevel Modeling

Toward a New Paradigm of Conceptual Modeling and Information Systems Design

The paper presents a novel approach to conceptual modeling that serves to address a fundamental conflict inherent in designing languages and information systems. It not only promises to improve the economics of developing and using models and respective software systems, but also to foster the empowerment of users by providing them with specific languages that correspond directly to the perspective on a domain they are used to.

DOI 10.1007/s12599-014-0350-4

The Author

Prof. Dr. Ulrich Frank (✉)
Chair of Information Systems and Enterprise Modeling
Institute for Computer Science and Business Information Systems
University of Duisburg-Essen
Universitaetsstr. 9
45141 Essen
Germany
ulrich.frank@uni-due.de

Received: 2013-10-13
Accepted: 2014-04-23
Accepted after two revisions by Prof. Dr. Karagiannis.

This article is also available in German in print and via http://www.wirtschaftsinformatik.de: Frank U, (2014) Mehrebenen-Modellierung. Ein Vorschlag für ein neues Paradigma der konzeptuellen Modellierung und des Entwurfs von Informationssystemen. WIRTSCHAFTSINFORMATIK. doi: 10.1007/s11576-014-0438-y.

1 Introduction

It is widely accepted that the construction of information systems calls for the development of conceptual models. However, the prospects of conceptual models (for an elaborate discussion of foundational terms such as model, conceptual model, modeling language, etcetera, see Mahr 2009; Frank 2011b) are offset by a number of challenges that compromise their beneficial use in practice.

Remarkable Effort The construction of comprehensive conceptual models requires a quantity of resources and time beyond the capability of many organizations.

Quality The utility of a conceptual model is dependent on its quality. However, developing high quality models requires a level of expertise and experience that is not available in many organizations.

Poor Protection of Investment Often, the use of conceptual models is restricted to the analysis and design phases of software systems. Subsequent modifications are frequently restricted to code and neglect to synchronize the affected conceptual models, with the consequence that models are devalued.

Widely Restricted to Experts Conceptual models are intended to foster communication among various groups of stakeholders, including those lacking specific training in modeling. However, non-experts are often not keen to look into conceptual models, nor are they capable of designing them on their own.

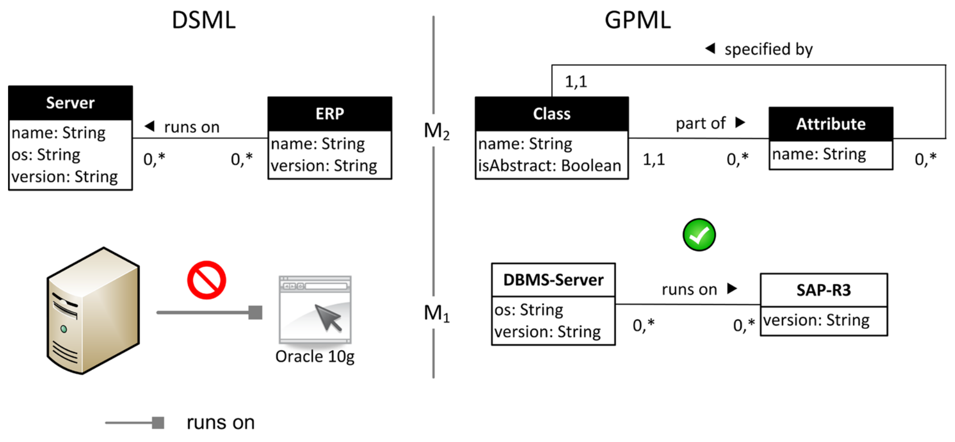
Various research activities have sought ways to address these problems. They include the construction and reuse of reference models (e.g., Fettke and Loos 2007), the use of models at runtime (e.g., Morin et al. 2009), and guidelines to promote model quality (e.g., Schütte 1998). With respect to the problems outlined above, one recent development is of particular relevance. Domain-specific modeling languages (DSMLs) are characterized not only by their claim to substantially improve the productivity of both modeling and software implementation, but also to promote model quality and user involvement. However, DSMLs are not the silver bullet for developing and maintaining information systems. Like conceptual modeling in general, DSMLs in particular face a fundamental conflict of system design that counters the benefit they can bring.

A thorough analysis of the challenges of system design provides a foundation on which the current research builds to offer a novel approach to creating DSMLs and their respective tools. That approach enables a substantial diminishing of the fundamental design conflict and has the potential to get more people actively involved in modeling. This approach took some time to evolve. On the one hand, it required a paradigm shift in the sense of conceding, or at least relaxing, the interpretation of certain principles that we have believed in and propagated for many years. On the other hand, it seemed at first to be impossible to implement the required tools in a satisfactory way due to the inherent limitations of prevalent programming languages.

This paper is structured as follows: first, the problem that motivated the proposed approach is elucidated by considering benefits, challenges, and conflicts related to the design of DSMLs. Then, the prospects for and challenges of multilevel modeling are elucidated,

Published online: 06 November 2014

Fig. 1 Illustrating the benefits of DSMLs over GPMLs



and based on that information, its attendant requirements are described. Against this background, a recursive language architecture and the conceptual foundation for the corresponding (meta) modeling environment are introduced. Subsequently, the benefits of the approach are demonstrated with an exemplary use scenario followed by a critical evaluation of the presented approach and concluding remarks.

2 Domain-Specific Modeling Languages

Language is a constituent of the construction of information systems. Language is however not only crucial for implementing software, but is also essential for designing and using software. The only way we can make sense of software, whether during the build or run time, is through human language, not through machine language. Therefore, not only the construction of a system but also the interaction with a running system depends on the linguistic representation of the system. However, not every language is appropriate for representing these artifacts.

2.1 Promises

Many system designers will regard a general-purpose modeling language (GPML), such as the UML, to be the instrument of choice for developing conceptual models. However, in everyday life we would regard it as entirely unreasonable if we were told to restrict our communication to a language with just a few primitive concepts such as *class* or *attribute*. Instead, we expect a language to provide concepts that support elaborate communication without forcing us to explain everything from scratch.

In recent years, this thought has led to the development of modeling languages designed for specific domains. A DSML is a modeling language that is intended to be used in a certain domain of discourse. It is based on concepts that were reconstructed from technical terms used in the respective domain. DSMLs promise various advantages over GPMLs (see the illustration in **Fig. 1**). First, they promote modeling productivity by freeing modelers from the need to reconstruct domain-level concepts from semantic primitives. At the same time, they foster system integrity, since inappropriate use of domain-specific language concepts is prevented to a certain degree by the abstract syntax and semantics of a DSML. Furthermore, DSMLs promise to provide a better medium for communicating with prospective users: On the one hand, their concepts are directly related to the technical terms prospective users are familiar with. On the other hand, they feature a specific concrete syntax – usually, but not necessarily, a graphical notation – that also fosters comprehensibility of models. In recent years, the same rationale driving the development of DSMLs has led to the construction of domain-specific programming languages (DSPLs). DSPLs provide programmers with domain-specific concepts, thereby increasing productivity and fostering software integrity. In an ideal case, models specified in a DSML are mapped to the code of a corresponding DSPL.

2.2 Challenges

The conflict between the benefits and drawbacks of semantics constitutes a fundamental challenge for information systems design in general. It is not especially relevant to the design of GPMLs, since they are by definition characterized by concepts on a low level of semantics that leave room for broad interpretation. However, the semantics challenge does affect the design of a DSML. The more specific its concepts, i.e. the more semantics they include, the higher the potential productivity gain in those cases where it fits. On the other hand, the more generic the concepts of a DSML are, the wider its range of reuse and, hence, its economies of scale. While it is hardly possible to calculate the optimal level of semantics a DSML should have, it is nevertheless required to make corresponding decisions, specifically to determine whether or not to develop a DSML for a certain domain and how the scope of the domain should be defined. Some authors suggest avoiding the issue by focusing on just one particular organization, and typically use one or more of three arguments to justify that recommendation. First, there is the quasi ontological hypothesis that every organization is characterized by idiosyncratic peculiarities that hinder the reuse of concepts from other organizations. The second argument relates to competitiveness: If a DSML is regarded as an asset that generates competitive advantage, it makes sense not to make it available to competitors (Kelly and Tolvanen 2008). Third, there may be no need for cross-organizational reuse if using a DSML in only one organization offers clear economic benefits. Kelly and Tolvanen strongly support the third argument (Kelly and Tolvanen 2008, p. xiii), and others adopt a similar position (Völter 2013; Fowler 2011). Without further differentiation, such as of an organization's mission, the qualifications of its employees, and its specific experience with modeling tools, it seems rash to unconditionally support those statements. All three arguments that support the restriction of

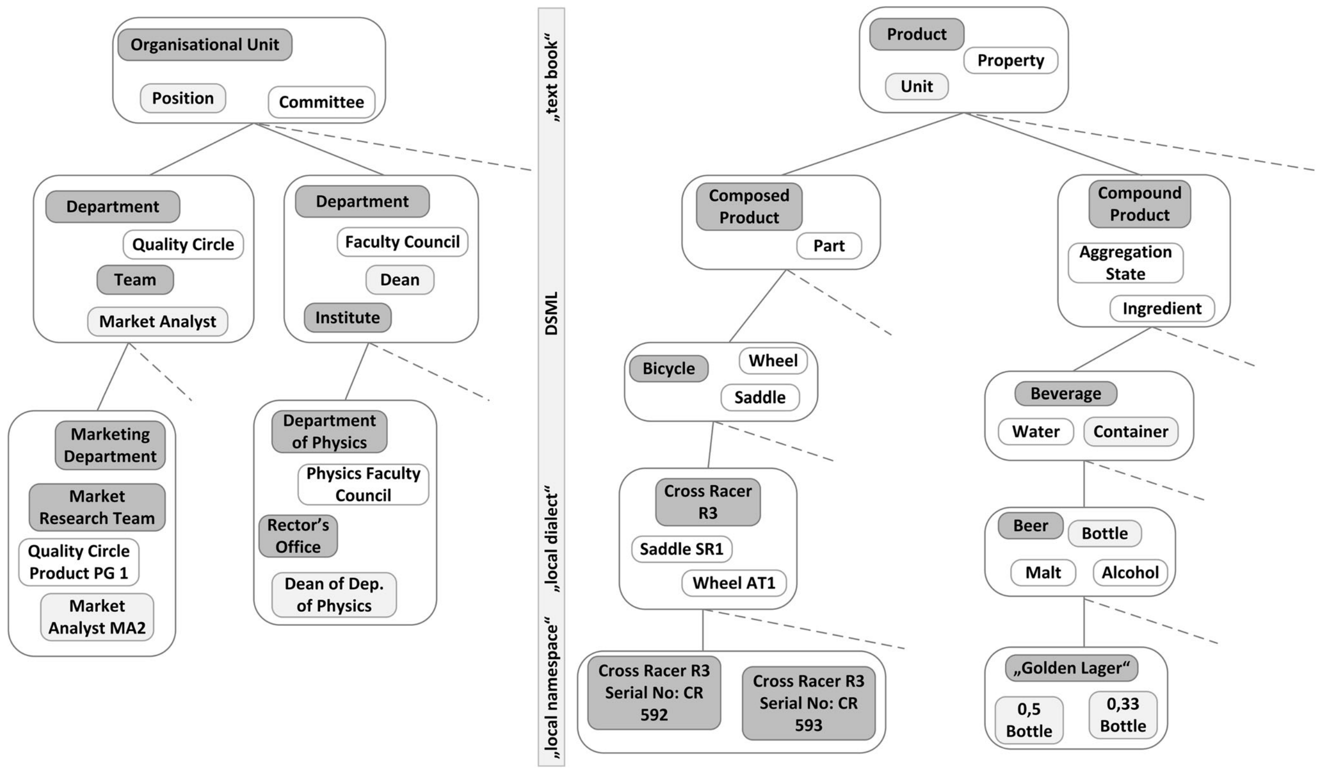


Fig. 2 Illustration of exemplary language hierarchies

a DSML to one organization can be countered. While the ontological assumption certainly seems plausible, there are two reasons why it should not be taken at face value. First, there is remarkable evidence that the action systems of an entire range of organizations can successfully use the same concepts, and examples can be seen in the widespread use of ERP systems and office applications. Second, the actual inter-organizational variance in the use of concepts may be substantially the result of an evolution that is characterized by contingency and chance. Hence, variance is not an inevitable ontological phenomenon. Third, it is conceivable that various organizations, whether competitors or not, can use a common DSML without compromising their competitiveness. Again, the dissemination of so-called standard ERP systems offers strong evidence for this argument.

3 A Multilevel Approach to Conceptual Modeling: Prospects and Requirements

The approach that this paper proposes to promote the construction of more versatile DSMLs is based on the introduction of multiple modeling levels. It is presented in three steps. First, the general idea is demonstrated. Second, requirements and challenges related to realizing multilevel modeling are analyzed. Third, the language architecture that facilitates multilevel modeling is presented.

3.1 Background: The Common (Re-)Use of Technical Languages

Exploiting the potential of DSMLs requires diminishing the design conflict that we considered above. This is possible by developing languages that enable both a wide range of reuse and specific support for particular cases. The actual use of technical languages offers clues as to how this demand could be satisfied. It is a major characteristic of modern societies that they are built on a large variety of technical languages. These technical languages often form hierarchies. At the top, there are reference languages that address entire domains independent of the peculiarities of particular use cases. These are typically used in textbooks, and as a result the textbook terms are intentionally kept at a relatively high level of abstraction, allowing for a certain range of interpretations. Reference technical languages are adapted to specific domains, that is, to the domain of a certain industry or a large organization. More specific technical languages can be introduced at multiple levels, so a technical language might be applied industry-wide or it might be organization specific or at a lower level, the technical language might be the jargon used within particular units or projects. The simplified examples in **Fig. 2** illustrate this kind of language hierarchy. The example on the left represents concepts of languages for describing organizational structures. The example on the right shows concepts used to describe products on different levels of abstraction.

The shades of gray serve to indicate how concepts are refined on a lower level. The term *Organizational Unit*, for instance, is refined into *Department*, *Team*, etcetera, while *Department* may be further refined to *Marketing Department*. Hence, a concept on a certain level is reused on all lower levels that refine it. Therefore, these hierarchies of technical languages promote both economies of scale, through extensive reuse of "textbook" level concepts, and productivity, by offering more specific concepts on the level of "local dialects." Using them as a model for creating hierarchies of DSMLs would suggest that common textbook level or reference DSMLs should be designed by experts who possess deep

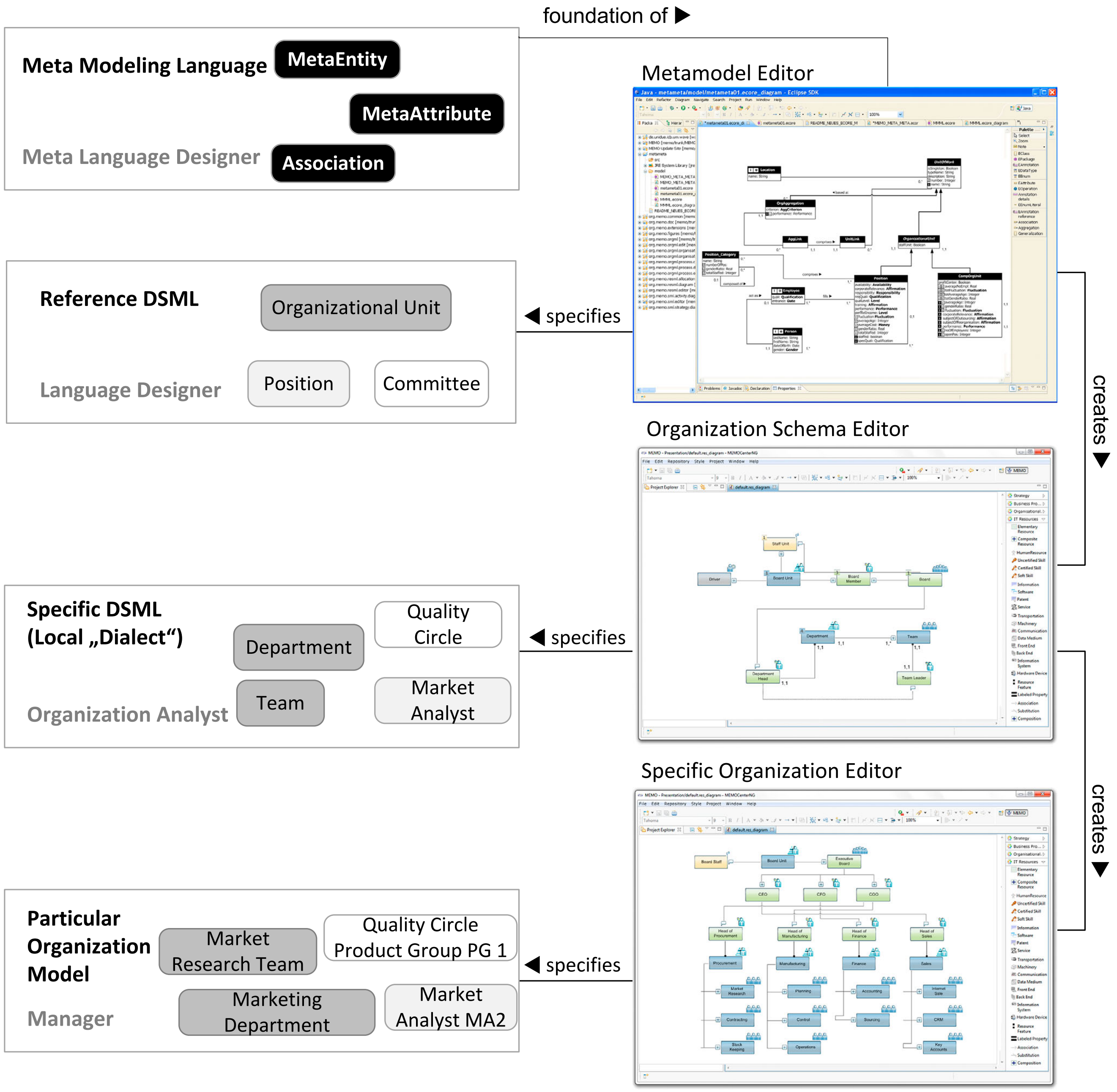


Fig. 3 Illustration of multilevel modeling with two DSMLs

knowledge about the general domain and have rich experience with designing DSMLs. More specific or "local" DSMLs could be designed by local domain experts using the common DSML. The more specific a DSML, the less expertise is required to design it. **Figure** 3 illustrates this idea of multilevel DSMLs, and multilevel modeling in general, using an example with three levels of classification. Each editor above M_0 not only enables the design of (meta) models, but also facilitates the creation of more specific editors, for example, by generating corresponding code.

In order to diminish the essential conflict inherent in designing DSMLs, and hence to allow for both a wide range of reuse and high productivity gains in particular cases, a multilevel modeling approach has to aim to minimize conceptual redundancy across all levels of abstraction. To achieve this, the following proposition has to hold for all levels i for $i \geq 1$: Every concept that is shared by all intended application domains on level i should already be specified on $i + 1$. Otherwise, the same specification would have to be repeated for all more specific language definitions, resulting in redundancy or, in other words, unsatisfactory reuse. Note that this is a formal orientation only. It does not solve the problem of appropriately tailoring domains, nor does it allow determination of the adequate number of classification levels (see Sect. 7).

3.2 Requirements

Applying the idea of multilevel technical languages to the construction and use of DSMLs as outlined in **Fig.** 2 seems to diminish the conflict inherent in designing DSMLs. However, the examples in **Fig.** 3 reveal that the realization of a multilevel modeling environment is a far from trivial matter. Doing so means tackling serious obstacles that relate to established principles both of conceptual modeling and of prevalent programming languages. Therefore, designing a multilevel modeling system requires nothing less than challenging the dominant paradigm. Our analysis of the corresponding requirements that evolved from our long-standing investigation of DSMLs and respective tools is structured as follows. First, we will focus on requirements related to principles of conceptual modeling and to respective language architectures (RLA). Second, we will look at requirements related to the limitations of traditional programming languages, because the economic realization of multilevel modeling depends on the implementation of corresponding tools (RI). Finally, we will focus on requirements aiming to bridge the gap between the prevalent paradigm and the proposed approach (RB).

RLA-1: Flexible Number of Classification Levels There is a need for a language architecture that allows for an arbitrary number of classification levels. The rationale for that is illustrated by the examples in **Fig.** 2 indicating that the number of language levels seems to be variable. While three levels are sometimes sufficient, in other cases four or more levels may be more appropriate. Traditional language architectures such as the MOF (Object Management Group 2006) are not satisfactory in this respect, since they are characterized by a fixed number of classification levels.

In traditional language architectures, instantiation and specialization are mutually exclusive (Frank 2012a): If class A is an instance of metaclass B, then A cannot be a subclass of B at the same time. The examples in **Fig.** 2 illustrate the well-known fact that in natural languages it is often not obvious whether we are dealing with an instantiation or a specialization relationship. We might ask, for example, whether a particular brand of beer is an instance or a specialization of *Beer*. However, we cannot be certain that even the most systematic examination of the two concepts will permit a clear decision to be made. To offer another example, the metaclass *Organizational Unit* on the *textbook* level may include attributes such as *number of employees* or *performance*. A class such as *Department* on the level below should have the same attributes. Specializing *Department* from *Organizational Unit* would produce exactly this effect. At the same time, the *Organizational Unit* may comprise attributes such as *name* or *is permanent*, which are intended to be instantiated on the level below. This perspective would recommend an instantiation relationship, which would be in clear conflict with the previous choice. These considerations lead to the following requirement:

RLA-2: Relaxing the Rigid Instantiation/Specialization Dichotomy There is a need for concepts that allow the reuse enabled by instantiation to be combined with that promoted by specialization, but without jeopardizing a system's integrity. The rationale is that the postulate of economic specification implies a need to specify properties of concepts that are assigned to a certain level on a higher level already, regardless of whether they are reused through instantiation or specialization. As the above *Organizational Unit* example shows, a combination of specialization and instantiation works in natural language.

In contrast to traditional language architectures, the example in **Fig.** 2 indicates that a certain language level is not restricted to classes on the same classification level. For example, in addition to concepts such as *Organizational Unit* or *Position*, a textbook may include a concept such as *Employee*. While *Organizational Unit* could be represented by a metaclass on M_2, *Employee* may correspond to a class on M_1.

RLA-3: No Strict Separation of Language Levels Multilevel modeling requires a versatile conception of (meta) models that allow classes on different classification levels to be part of one model. The rationale behind that is that hierarchies of natural technical languages indicate that concepts of a particular language in a hierarchy are not necessarily restricted to a certain classification level. In other words, in technical discourses there are useful sentences that contain concepts on different classification levels. For example, "Cross Racer R3 is one of our most successful products." While "Cross Racer R3" may represent a concept on M_1, *Product* may represent a concept on a clearly higher level of abstraction (see the scenario in Sect. 5).

A tool environment is of crucial relevance for a multilevel modeling approach. This is not only because modeling tools foster the economic creation, analysis and maintenance of models. In addition to that, tools are required to generate editors from the specification of DSMLs.

RI-1: Straightforward Representation of Language Architecture The hierarchy of DSMLs that is enabled by the outlined architecture of modeling languages needs to be mapped somehow to the internal representation of corresponding tools. This includes not only mapping classes, attributes and the like but also classes on arbitrary classification levels, and concepts that address RLA-2 and RLA-3. Ideally, the language used to implement model editors includes concepts that are semantically equivalent to those defined by the respective (meta) modeling languages. The rationale for this is that the reconstruction of modeling concepts with programming languages demands substantial effort and threatens integrity.

If it is not possible to represent the entire language hierarchy in the tool environment, a model hierarchy needs to be distributed to editors that operate on a particular part of the language hierarchy only. In this case, the following two requirements have to be taken into account.

RI-2: Cross-Level Integrity The tool environment needs to effectively support the integrity of a hierarchy of DSMLs. As a consequence, the modification of a model on level n has to be consistently propagated to all affected models on all levels m for $m < n$. Rationale: The appropriate use of a model requires accounting for updates of modeling concepts.

RI-3: Cross-Level Navigation An environment for multilevel modeling should allow navigation through a hierarchy of (meta) models. Rationale: On the one hand, the use of a local DSML may create the need for an explanation of a concept of a higher level DSML. On the other hand, studying a metamodel may generate a desire to inspect examples of corresponding models.

RI-4: Support for the Convenient Creation of Model Editors A reference DSML may be the root of a large tree of more specific DSMLs. Specific DSMLs are specified by local domain experts. The efficient use of a more specific DSML depends on corresponding model editors that can be created with little effort. Ideally, a meta-modeling environment should allow the generation of a model editor to a wide extent from a metamodel of a DSML and an additional specification of the concrete syntax. Rationale: Users who may be able to specify a local DSML cannot be expected to implement a corresponding editor.

Satisfying the above requirements requires overcoming the current paradigm of conceptual modeling. However, that does not necessarily mean seeking disruptive change. Instead, it seems more appropriate to avoid giving up beneficial aspects of the current paradigm and preserve existing assets, which leads to the final two requirements.

RB-1: Clear Specification of Classification Levels Requirement RLA-3 implies that a model may include classes on different levels of classification. As a consequence, the classification level of a class cannot be concluded from the model it is part of. The language architecture should provide concepts that allow the explicit definition of the classification level of every class. They should also be represented in the accompanying tool environment. The rationale behind this is that if a class is to be interpreted appropriately, it is essential to know which classification level it is supposed to represent. There is, for example, a clear semantic difference between a class *Document* on M_1 and a metaclass *Document* on M_2.

RB-2: Backward Compatibility The language architecture should facilitate the integration of existing (meta) models. For this purpose, the meta-modeling language has to include concepts that correspond to those used in existing meta-modeling languages. The rationale here is that there is already a considerable number of DSMLs in the area of enterprise modeling. To protect these assets, the effort required to port them to a multi-level language architecture must not be prohibitive.

4 Language Architecture and the (Meta) Modeling Environment

Our earlier attempts to satisfy the above requirements using a MOF-like language architecture resulted in adding concepts to a meta-modeling language that served as workarounds (Frank 2011a). The corresponding modeling tools were implemented in a traditional programming language that features only one classification level. Therefore, the representation of multiple levels of classification was possible only by overloading either M_0 or M_1. As a consequence, the conceptual mismatch made it impossible to convincingly address requirements RI-1 to RI-3. Other meta-modeling environments such as MetaEdit (Kelly et al. 2013) or ADOxx (Fill and Karagiannis 2013), though mature and powerful, do not support multiple classification levels either. With respect to these serious limitations, we decided upon a paradigm shift. Both the language architecture and the programming language used to implement the tool environment should be based on a recursive construction permitting an arbitrary number of classification levels.

4.1 Background: "Golden Braid" and XMF

The idea of recursive language architectures has arguably become popular through the "golden braid" metaphor that Douglas Hofstadter used for his sophisticated praise of recursion (Hofstadter 1979). It is based on the idea that a class can be regarded as also being an object that is instantiated from a metaclass, which in turn can be seen as an object instantiated from a higher level class (i.e., "everything is an object"). To avoid a *regressus ad infinitum*, the instantiation process is recursive (see the description below), that is, a core (meta) class is instantiated from itself (for a more detailed description see Clark et al. 2008a, 2008b).

Very few programming languages are based on a golden braid architecture (e.g., the object-oriented extensions of Lisp, Smalltalk, and Ruby). Among these languages, Smalltalk is especially appealing. It treats classes as objects and is available in powerful development environments. Unfortunately, Smalltalk is not suited for our purpose since it does not allow the definition of metaclasses above M_2. Furthermore, it does not feature metaclasses as classes of a set of classes. Each class is assigned precisely one default metaclass, and a metaclass in turn has only one instance. XMF (executable Metamodeling Facility) is not limited by this constraint (Clark et al. 2008a, 2008b). It is a language execution engine featuring a metamodel called XCore (Clark et al. 2008a, p. 43). Every language that is specified in XCore can be executed by XMF. XMF allows access to and modification of its own specification and its runtime system. Hence, there is no clear distinction between the language and a corresponding meta-language, and therefore XMF is reflective. Furthermore, it includes tools for building compilers for further languages. That makes XMF a meta-programming facility that allows execution of code written in different languages in the same runtime system. Apart from the fact that XCore features a golden braid architecture, there are further two reasons for choosing XMF as an implementation language. It offers most of the properties essential to a meta-modeling language (RI-1, RB-2) and also permits the modification of XCore to satisfy the specific requirements of designing and using DSMLs.

The golden braid architecture in general, and XCore in particular, is based on concepts that seem to violate acknowledged principles of meta-modeling and that therefore can cause confusion. XCore makes use of a circular relationship in that the central metaclass *Class*, which is amongst others associated with a meta-attribute, a meta-operation, and a meta-association (see **Fig. 4**), is an instance of itself. At the same time, *Class* inherits from *Object*. Hence, every class is an object and can be executed. *Object* is itself instantiated from *Class*. Furthermore, every instance of *Class* can inherit from every other instance of *Class* as long as cyclic relationships are avoided. The lean recursive structure of XCore enables the creation of an arbitrary number of classification levels, although not without effort. Initially, *Class* is located on M_2. If a metaclass on a higher level of classification is required, one will first instantiate a class from *Class*, which will result in a class on M_1. Subsequently, one would have the new class inherit from *Class*, which would lift it up to M_2, because it would inherit the instantiation capability of the *Class*. Instantiating the new class and having the resulting instance inherit from the original metaclass *Class* would result in lifting the instance up to M_2 and, in consequence, its classification level, which was previously M_2, up to M_3. Since the lifting procedure can be repeated indefinitely, language hierarchies with any number of classification levels can be realized.

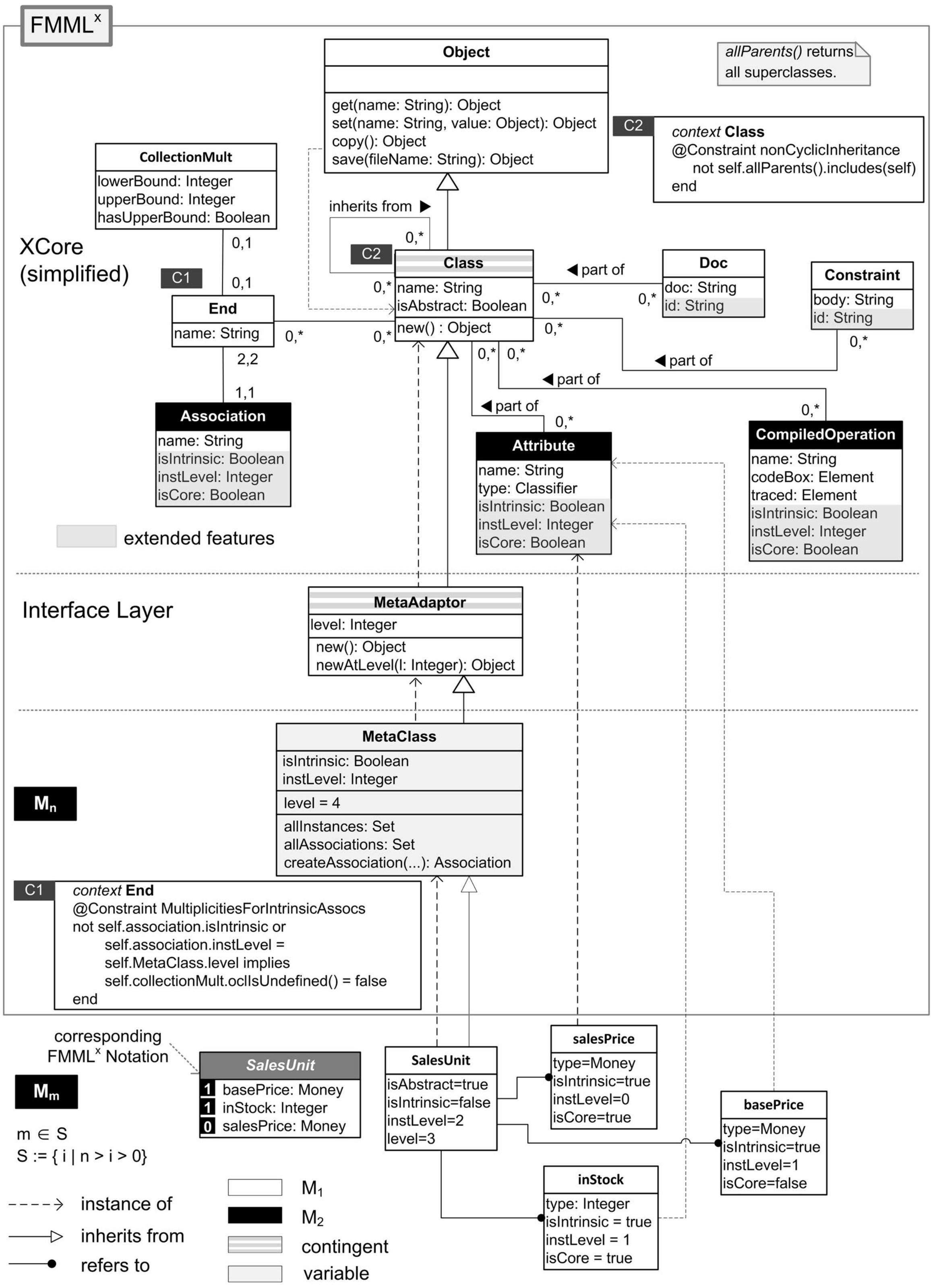


Fig. 4 Excerpt of the FMML^x metamodel and its relation to XCore

4.2 Conceptual Foundation: Flexible Meta-Modeling and Execution Language

During the design of the modeling language architecture and the corresponding implementation of prototypical modeling tools, an especially appealing design option emerged. If a common representation of (meta) models and code could be accomplished, there would no longer be a need to transform (meta) models into code, and to synchronize models and code later on after changes have occurred. Hence, models would be executable, that is, offering operations accepting of queries and changes to their state. A straightforward approach would be to use XCore directly as a common foundation. However, that would not be satisfactory for the following reasons: First, the creation of language hierarchies is too cumbersome (see the description above). Second, XCore does not account for an explicit classification level. In fact, the classification level of a class may even be contingent, that is, a class can be instantiated into a class on M_n and into another class on M_m with $m \neq n$ at the same time. In other words, a class may be level-agnostic. While this feature offers outstanding flexibility, it is not satisfactory from a conceptual point of view (RB-1). Third, XCore does not provide direct support for dissolving the instantiation/specialization dichotomy (RLA-2). To address these shortcomings, the design was aimed at modifying XCore accordingly. Furthermore, a complementary graphical notation was created to foster the convenient use of specific language concepts. We call the resulting meta-modeling language Flexible Meta-Modeling and Execution Language (FMML^x), where the "x" is intended both to express the flexible classification level of the metamodel and to indicate that the metamodel, as well as all models instantiated from it, are executable within XMF. In the following, the extensions applied to XCore are presented with reference to the representation of the FMML^x metamodel in **Fig. 4**. Note that only key properties of the metamodel are accounted for. Most of the operations and constraints are omitted. **Figure 4** also includes an example of classes specified with FMML^x. To avoid misinterpretation, the classification level of the represented concepts is indicated by the background color of each class name (see the legend of **Fig. 4**). While *contingent* means that a class can represent different levels at the same time, *variable* indicates that its level can be explicitly modified. This may seem confusing at first. However, the respective concepts should become clearer with the illustration of their application in the subsequent examples.

Explicit Classification Level and Convenient Construction of Classes on Arbitrary Classification Levels For this purpose, the auxiliary metaclass *MetaAdaptor* was introduced. It is instantiated from *Class* and inherits from *Class*. First, the attribute *level:Integer* was added. The core class of FMML^x, *MetaClass* is instantiated from *MetaAdaptor* and inherits from it. The intended classification level of *MetaClass* can be easily defined by initializing the attribute *level:Integer*. Furthermore, all classes that are instantiated from *MetaClass* or from one of its instances, also inherit from *MetaClass*. Hence, all (meta) classes in a multi-level modeling system both inherit the attribute *level:Integer* and instantiate it. Whenever a class or object is instantiated, its *level* attribute is initialized with the intended number. The default level is $n - 1$ if the class it is instantiated from is located on n. For this purpose, the instantiation operation *new*() inherited from *Class* was overridden. To enable the creation of classes on levels lower than $n - 1$, a further instantiation method, *newAtLevel* (*l: Integer*) was added to *MetaAdaptor*. In **Fig. 4**, the contingent classification level of *Class* and *MetaAdaptor* is represented by a striped bar. Note that users of FMML^x would normally see only *MetaClass* and its inherited properties. The classification level of *MetaClass* can be defined according to specific needs. In the example in **Fig. 4**, it is set to 4.

Relaxing the Rigid Instantiation/Specialization Dichotomy The problem that is addressed by requirement RLA-2 has been known for some time. For the extension of XCore we used a modification of "intrinsic features" (Frank 2011a). They are similar to "powertypes" (Odell 1994) and especially to "deep instantiation" (Atkinson and Kühne 2008). However, unlike powertypes and deep instantiation, intrinsic features comprise not only attributes but also operations and associations. Furthermore, it is possible to specify an entire class as intrinsic, which implies that all its features are intrinsic. Intrinsic features enable what one could call selective specialization or deferred instantiation: A feature that is marked as intrinsic can be instantiated only on the specified instantiation level. To implement the intrinsic features of FMML^x, the implementation of the meta-attribute (*Attribute*), meta-operation (*CompiledOperation*), and meta-association (*Association*) in XCore were modified. This is relatively easy since XMF treats attributes, operations, and associations as objects, the features of which can be defined through corresponding metaclasses. It was only necessary to add two attributes to these metaclasses: *isIntrinsic: Boolean* indicates if the respective feature is intrinsic, and *instLevel: Integer* allows the specification of the level where a proper instance would be located. These additional attributes are marked with a gray background in **Fig. 4**. To complete the implementation of intrinsic features, the *new*() operation defined in *MetaAdaptor* had to be further modified. It checks the intended instantiation level of intrinsic attributes, operations, or associations. Only if the instantiation level corresponds to the classification level of the respective class minus one will *new*() instantiate them. To illustrate the use of intrinsic features, we refer to the example used to motivate RLA-2. The attribute *numberOfEmployees: Integer* within *OrganizationalUnit* on M_2 would be marked as intrinsic (*isIntrinsic* $=$ *true*) and its intended instantiation level would be set to 0 (*instLevel* $= 0$). Instantiating *OrganizationalUnit* into *Department* would now result in inheriting *numberOfEmployees: Integer* to *Department*. The attribute would only be instantiated on the level below, for example, with the object that represents a particular marketing department. The FMML^x metamodel is supplemented by various executable constraints that are specified in XOCL, a variant of the OCL. Examples of XOCL constraints are shown in **Fig. 4**.

The recursive structure of the FMML^x enables considerable flexibility and reduces specification effort to a large degree. For example, a DSML used for modeling product types should allow the expression of associations between a product type and its components. Normally, this would require including a metamodel of associations in the specification of the DSML, even though a corresponding specification already exists for the meta-language with which the DSML was designed. FMML^x makes it possible to avoid this additional effort. By instantiating the metaclass *Product* from *MetaClass*, *Product* would at the same time inherit the specification/implementation of the meta-association that is part of FMML^x (where it is inherited from *Class* in XCore) as well as the specification/implementation of the meta-attribute (*Attribute*), the meta-operation (*CompiledOperation*), etcetera. By providing these basic language concepts on each level of abstraction above M_0, FMML^x enables the convenient extension of all languages of a particular hierarchy.

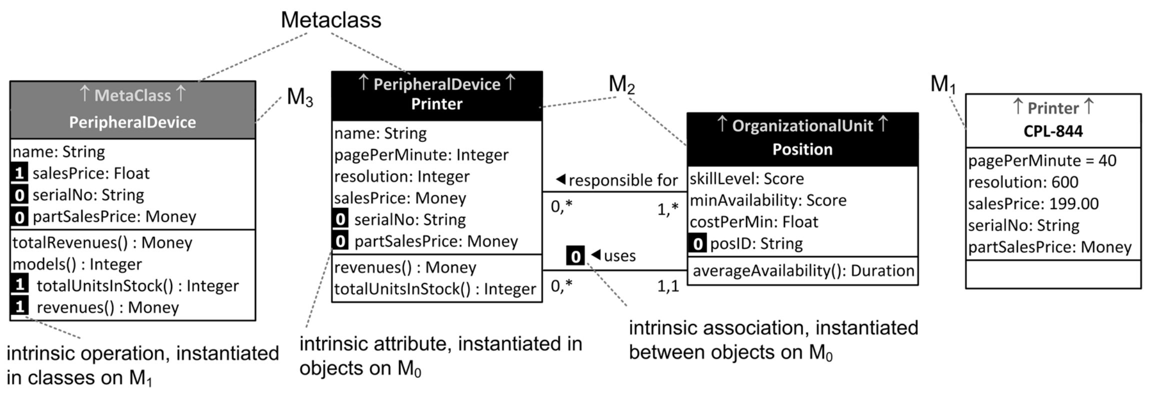


Fig. 5 Illustration of concrete syntax

FMML^x is complemented by a graphical notation. On the one hand, this facilitates the distinguishing of metamodels on different classification levels from object models using the style of the established UML class diagrams. On the other hand, it enables the representation of specific characteristics of FMML^x, such as intrinsic features. The notation is adapted from an existing meta-modeling language (Frank 2011a). The examples in **Fig. 5** illustrate key elements of FMML^x's concrete syntax. The color of the top bar that contains the name of a class serves to indicate the level of classification. Black represents M_2 and blue (shown as dark gray in this article) represents M_3. Further colors can be defined for higher levels of classification. Intrinsic features are marked by a white number in a black square. The number represents the level where the prospective instance of the feature is to be located. Note that this is the default notation of FMML^x. To enhance usability, it may be replaced by the specific concrete syntax of a particular DSML.

4.3 Meta-Modeling Environment

FMML^x is supplemented by a meta-modeling environment that extends Xmodeler, a meta-modeling framework that supplements XMF and that is implemented within the Eclipse framework. The framework has two main components. First, a generic model editor enables the creation of (meta) models by using a generic, UML-like notation. Second, a concrete syntax editor supports designing the symbols on which a graphical notation is based and additional widgets such as menus, listboxes, text editors, buttons, etcetera. The symbols created with this editor are mapped to their respective (meta) model elements using a generic tree structure that is part of Xmodeler, thereby completing the definition of a DSML. A model is represented by executable (meta) classes. They can be modified using either a graphical model editor or common software development tools such as browsers or editors. The model-view-controller pattern is used to synchronize a model and its diagram(s). Based on the specification of a metamodel, the definition of a corresponding notation and its respective mapping, Xmodeler facilitates generating a DSML editor.

The multilevel modeling environment builds on this framework. A FMML^x editor that was created with the generic components of the framework serves as a bootstrap editor. It is used to specify metamodels and corresponding notations of reference DSMLs and to generate corresponding editors (RI-4). Reference DSML editors serve to create more specific DSML editors, which in turn may be used for creating even more specific DSML editors. Finally, editors may be created that operate on objects on M_0. They do not allow for further instantiations or, therefore, for creating further model editors. **Figure 6** shows a model that illustrates the recursive architecture of the multilevel modeling environment. All model editors that constitute an integrated multilevel environment reuse the generic model editor that is part of the Xmodeler. Furthermore, each editor that is used to define further DSMLs makes use of the concrete syntax editor within the Xmodeler.

All (meta) models editors that belong to a certain hierarchy of DSMLs operate on models, that is, interrelated classes that are part of the same multilevel class hierarchy. Therefore, they facilitate navigation through the corresponding class hierarchy (RI-3). Furthermore, extensions of classes on a certain level are visible in affected lower level classes immediately. Adding an intrinsic feature triggers an update procedure that adds that intrinsic feature to all instances, and transitively to instances of instances, of the respective class. Propagating the deletion of a property on a higher level to affected lower levels cannot be entirely automated, because XMF does not feature static typing. Therefore, identifying elements on lower levels that are supposed to be deleted requires manual intervention. For example, assume a metaclass *Product* includes the operation ⟨*pricePerUnit:Money*⟩ that is available within its instances (i.e., classes representing particular product types). If this operation is deleted later on and there are other classes in the system that provide a method with the same interface, the lack of static typing prevents the straightforward identification of calling classes. Instead, additional inspection of possible calling classes would be required. Hence, requirement RI-2 (cross-level integrity) is supported to a large degree but cannot be entirely guaranteed by the tool environment. In order to promote both productivity and a consistent notation within a particular DSML hierarchy, the concrete syntax editor can reuse existing notations. It is possible to integrate editors of different DSML hierarchies into a multi-language editor by integrating the corresponding metamodels and concrete syntax specifications, thereby addressing requirements RI-2 and RI-3.

5 Exemplary Use Scenario

Even though the proposed language architecture and the modeling environ-

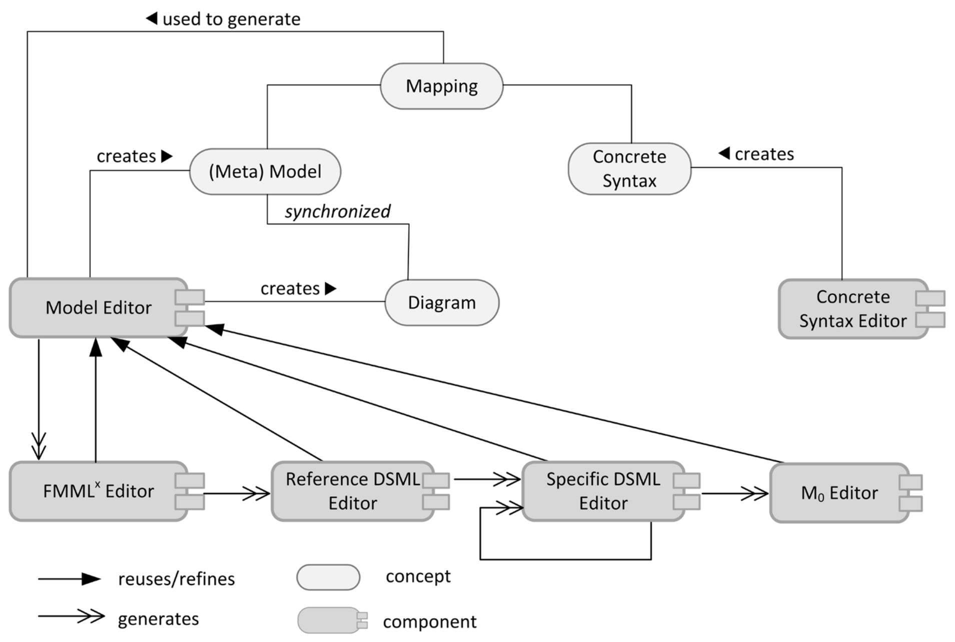


Fig. 6 Illustration of multilevel modeling environment

ment that supplements it are able to diminish the conflict inherent in designing DSMLs, the benefits of such an approach for creating and using DSMLs are not necessarily obvious. The following scenario, which corresponds to the example on the right-hand side of **Fig.** 2, is aimed at illustrating the potential not only of multilevel modeling, but also of using models as versatile representations that allow for interaction.

Products exist in a remarkable variety of types. Therefore, meaningfully representing a wide range of different products with just one class seems to be a hopeless undertaking. The diversity of product types is a specific challenge to those companies that offer a wide, ever changing range of different product types, such as department stores or e-commerce platforms. It creates a challenge, too, for software vendors that would like to reuse systems designed for certain product types to handle other product types, too. A further challenge for modeling products and their representation in information systems results from the fact that products and corresponding types that cannot be represented together on the only classification level that is available within traditional systems architectures (Frank 2002). Applying the conception of multilevel modeling to these challenges involves specifying a reference DSML for modeling products. The reference DSML would then be used to specify further specific DSMLs to model product types of certain categories. The simplified metamodel in **Fig.** 7 represents a generic conceptualization of products. The use of the background colors blue (shown as dark gray in the **Fig.** 7), black and white indicates that the metamodel includes classes on M_3, M_2, and M_1. The names printed in gray on top of metaclass names refer to the respective metaclasses. Note that since the classification level of *MetaClass* is variable, it can be instantiated into classes on different classification levels. In order to describe a product type in more detail, it will usually be required to refer to the parts from which it is made. These can be ingredients that are absorbed into a product or components a product is constructed from. The first case is represented in the metamodel as *CompoundProduct* and the second case as *ComposedProduct*. In both cases, the composite pattern is applied to describe how a product is made up of its constituents. Terms and conditions – which in the example are reduced to price – cannot always be directly assigned to a product because they may depend on the number of units or type of packaging. The abstract metaclass *SalesUnit* and its subclasses represent this notion. In cases where packaging is not relevant, the metaclass *BasicQuantity* serves to specify the units (e.g., liter, piece, etcetera) and the quantity to which the terms and conditions apply. The white digits in black rectangles mark intrinsic features. They indicate that a concept is supposed to be instantiated only on the classification level that is specified by the digit. For example, the subclasses of *Part* (M_3) need to be instantiated twice before the attribute *weight: Float*, which applies to product types, can be instantiated on M_1. Its attribute *serialNo:String* can only be instantiated on the M_0 level. The specification of ingredients requires a different classification level: An instance of *CompoundProduct*, which would be located on M_2, e.g., *Beer*, is assigned to an instanceof *Ingredient* on M_1. Therefore, *Ingredient* and its subclasses are specified on M_2 in the metamodel. Moreover, *Share* and *IngShare* are specified on M_1, because they are supposed to be instantiated only once, where their instances are linked to ingredient classes on M_1. The two exemplary constraints serve to prevent cyclic compositions on M_0 (C1) and cyclic specialization relationships (C2).

The example in **Fig.** 8 shows how a reference DSML is used to define a more specific DSML for modeling customized bicycles. It represents generic knowledge about the construction of bicycles, which is differentiated into the type and the instance level. For example, a certain type of bicycle may accept a range of different saddle types. However, for a particular instance of this bicycle type it is required to assign exactly one instance of a certain saddle type. The composition on the type level is represented by *composed_of* associations between the metaclass *CustomizedBicycle* and the metaclasses representing part types. The specific composition of a particular exemplar is represented by the *part_of* association. It applies only to the instance (M_0) level, which is indicated by the white "0" in a black rectangle next to the designator of the corresponding intrinsic association. The names printed in gray on

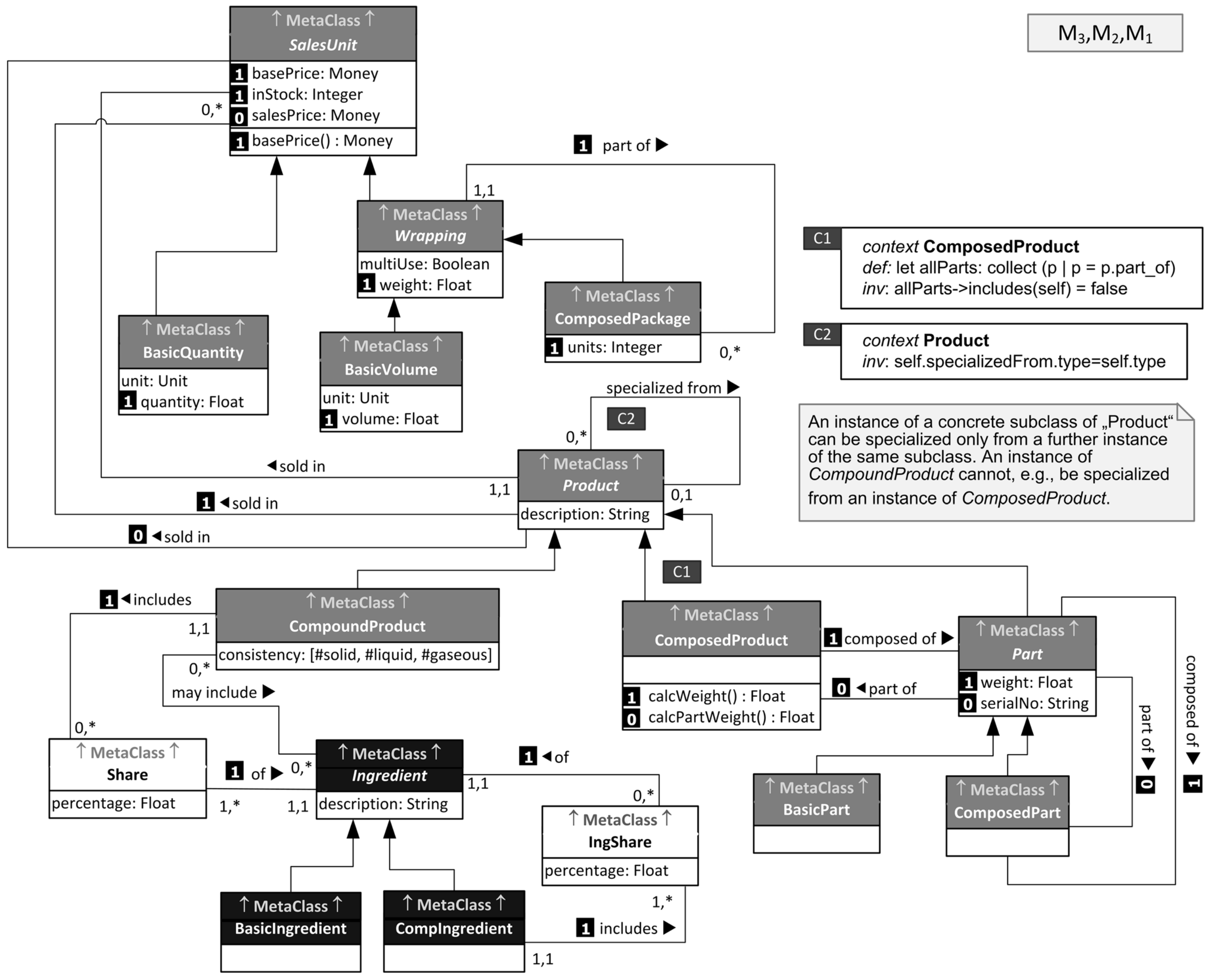


Fig. 7 Exemplary specification of a reference DSML for product modeling on M_3, M_2, M_1

top of metaclass names refer to the corresponding metaclass on M_3. Additional constraints allow the space of possible configurations on the M_1 level to be further restricted. For example, a type of carrier can be assigned only if a corresponding frame type can accept the mounting of a carrier.

Such a DSML could be provided for an entire industry, for example, to support software vendors who develop software for bicycle manufacturers or dealerships. It could also be used to model particular bicycle types. In addition to that, the DSML could be used to create models that guide the configuration of customized bicycles. These models could be specified for networks of dealerships or for a single dealer. **Figure** 9 shows a model of a certain type of customized bicycle that was specified using the DSML in **Fig.** 8. It can be regarded as a local DSML that guides and restricts the configuration of specific bicycles that correspond to a certain product type, in this case called *Customized-Racer_G5*. It includes all part types, instances of which can be used to build a particular customized bicycle. Additional constraints can be specified to exclude certain combinations.

Based on the model in M_1, a configuration tool can be built that helps a salesperson configure a particular bicycle. The mockup in **Fig.** 10 illustrates how such a tool might look. It could be integrated with stock management and accounting software, ideally by using respective DSMLs for these areas. Since it includes both representations of classes and references to particular instances, it combines the classification levels M_0 and M_1. Note that the example is based on the assumption that a particular customized bicycle is unique. One could also select a different approach, where each configuration is specified as a class, instances of which can be created later on. In that case, a further classification level would be required.

There is a clear difference between products like bicycles and beverages. While the identity of a bicycle will usually be preserved in an information system that represents it, one would not bother distinguishing representations of particular bottles. Therefore, it is impossible to use software that was designed for handling product exemplars with an identity of their own for products that lack this characteristic. However, multilevel modeling enables reuse of common concepts on a higher level of abstraction. The reference DSML in **Fig.** 7 can be reused for a wide range of product types. **Figure** 11 shows its application to beverages.

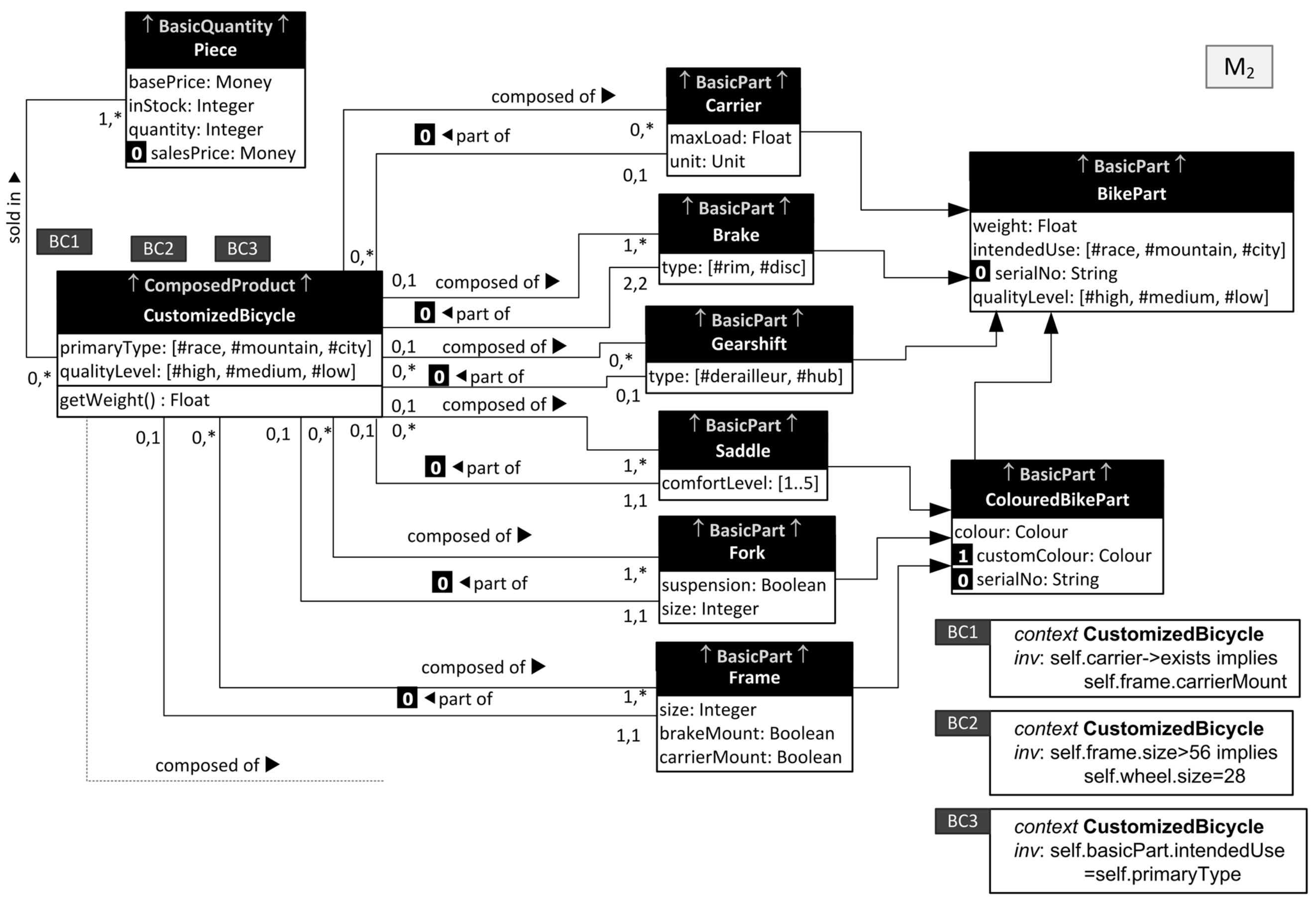


Fig. 8 Metamodel of DSML for specifying types of customized bicycles

The metamodel on M_2 specifies meta types of beverages: *Beer*, for instance, can be instantiated into specific brews on M_1. The metamodel represents knowledge about packaging by providing possible metaclasses of containers. For example, the metaclass *DisposableBottle* can be instantiated into a certain type of bottle, which could be defined by its volume and weight. In addition, possible ingredient types can be assigned to beverage metaclasses. In the example, all possible ingredients of beer are shown as instances of *Ingredient*. This prototypical instantiation corresponds to the known use of reference models and can be interpreted as a constraint that restricts the range of permissible instantiations of *Ingredient* assigned to a particular brew on M_1 (via an instance of *Share*). The respective constraint is also shown in **Fig.** 11.

Note that it may be an interesting ontological question whether *Water* (and other ingredients) should be modeled as a type (because it is clearly an abstraction from a particular physical occurrence and would allow for further specialization) or as an instance (because it hardly allows for further instantiation). In the example, we selected the first option. With respect to the subject of this paper it is important to stress that the model in **Fig.** 11 includes both metaclasses and classes.

While the example is restricted to beer and soft drinks, other metaclasses of beverages could be specified on this level, too. Note that for illustration purposes the example shows a simplified model that does not account for further useful concepts, which might, for example, capture commonalities of instances of *BasicVolume* or of instances of *CompoundProduct*. However, these details are not at the core of the proposed approach and would require extensive additional considerations. The mockup in **Fig.** 12 illustrates how a DSML could be presented to prospective users as an interactive application. The excerpt of a diagrammatic representation of the respective model illustrates its conceptual foundation. In a similar way to the previous example, it may have been generated from a respective metamodel and a corresponding definition of presentation elements. A type of beer is not further instantiated in an information system. Moreover, there is usually no need to distinguish particular bottles or containers. Therefore, the model in M_1 would not be instantiated any further. Nevertheless, the respective classes to represent a certain kind of beer in a certain sales unit could be assigned to an instance on M_0, such as, to an object representing a particular customer.

Note, however, that the number of classification levels may vary. While the above examples are based on four classification levels, it is conceivable to use more or fewer levels for certain product types.

6 Evaluation

The proposed approach to multilevel modeling is based on a language architecture that is in clear contrast to established principles of conceptual modeling and on an "exotic" meta-programming environment. Therefore, evaluating it is at present restricted to comparing it against requirements, discussing particular strengths and shortcomings, and relating it to similar work.

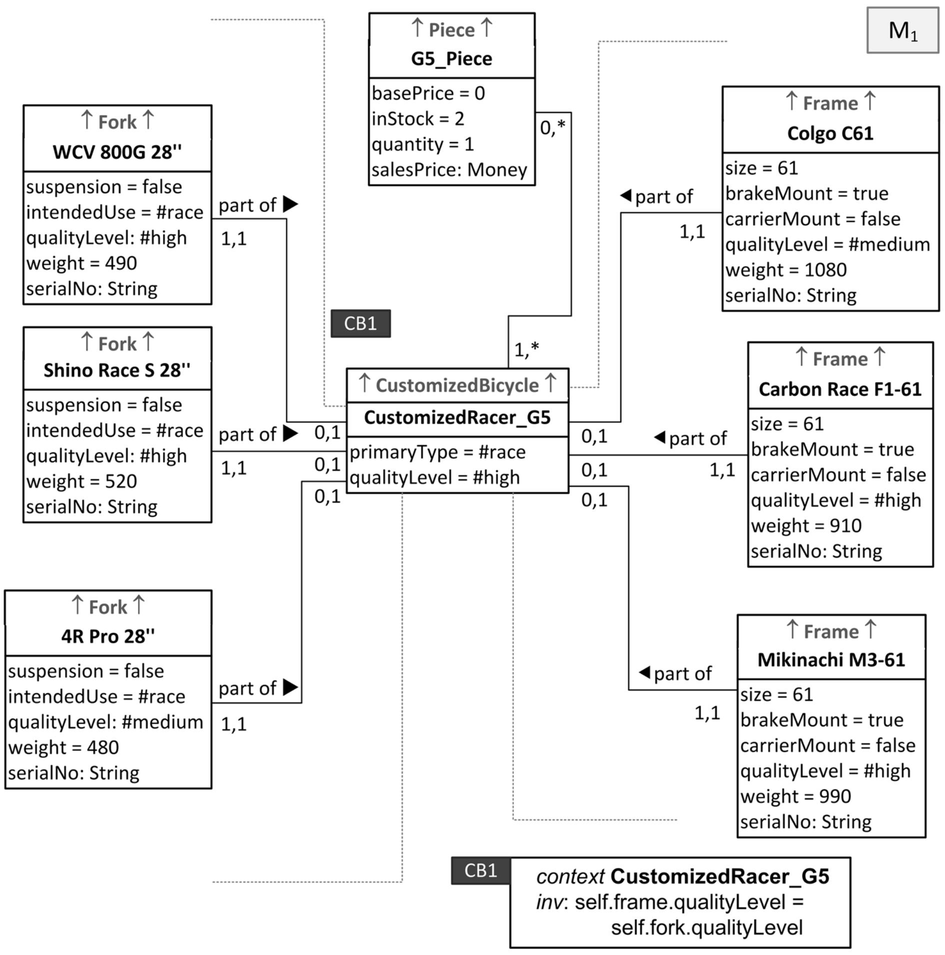


Fig. 9 Specific DSML for the configuration of bicycles

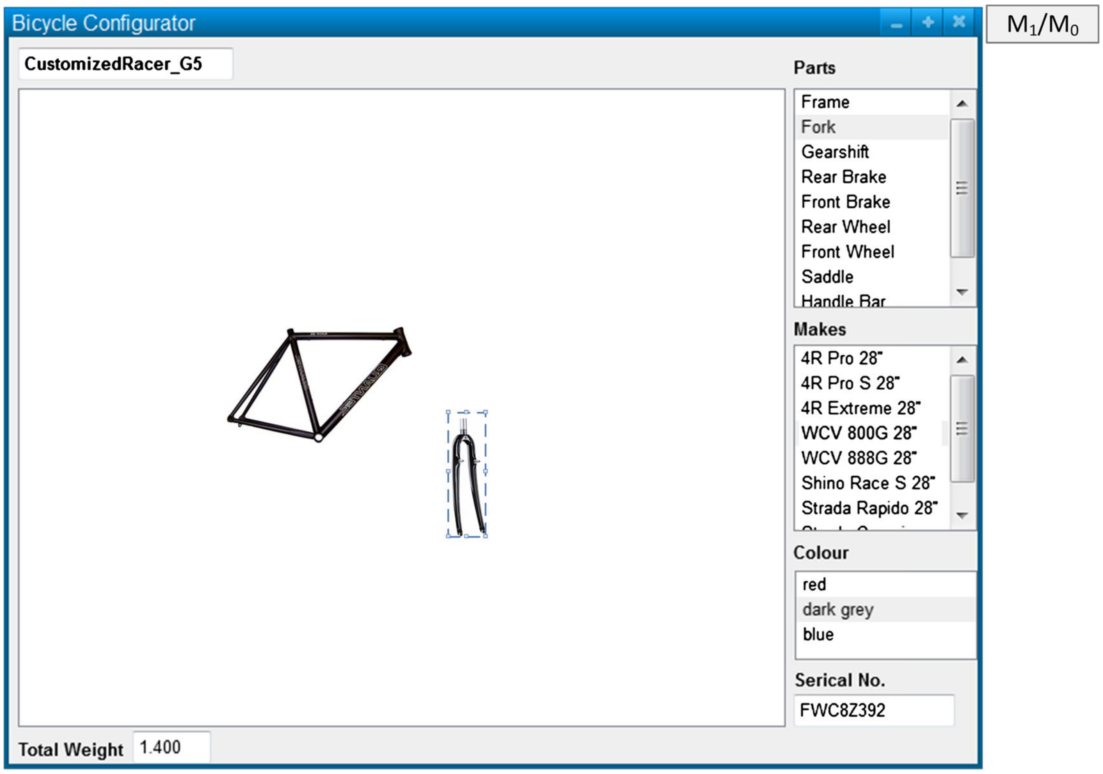


Fig. 10 Mockup of bicycle configuration on M_0 with references to M_1

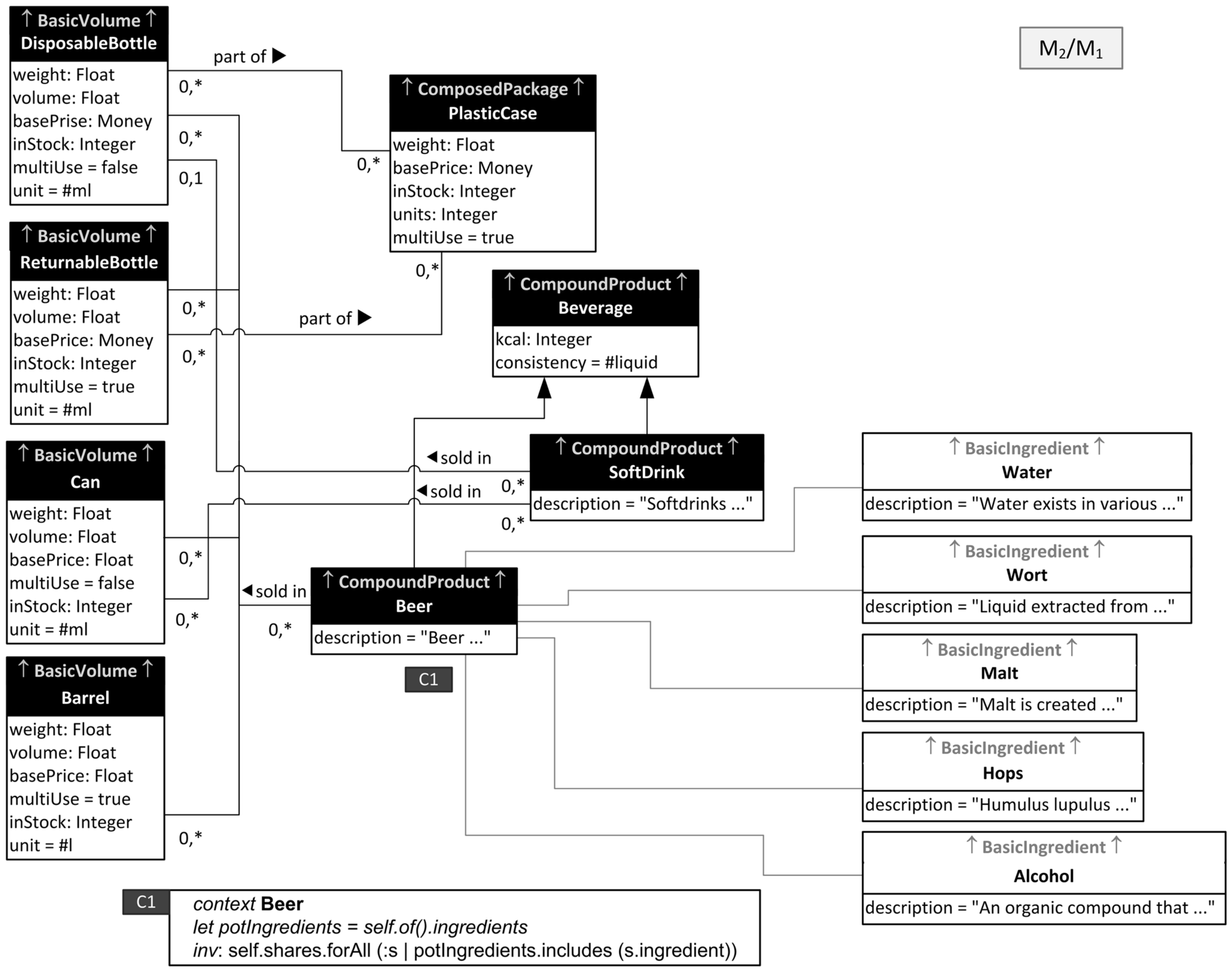
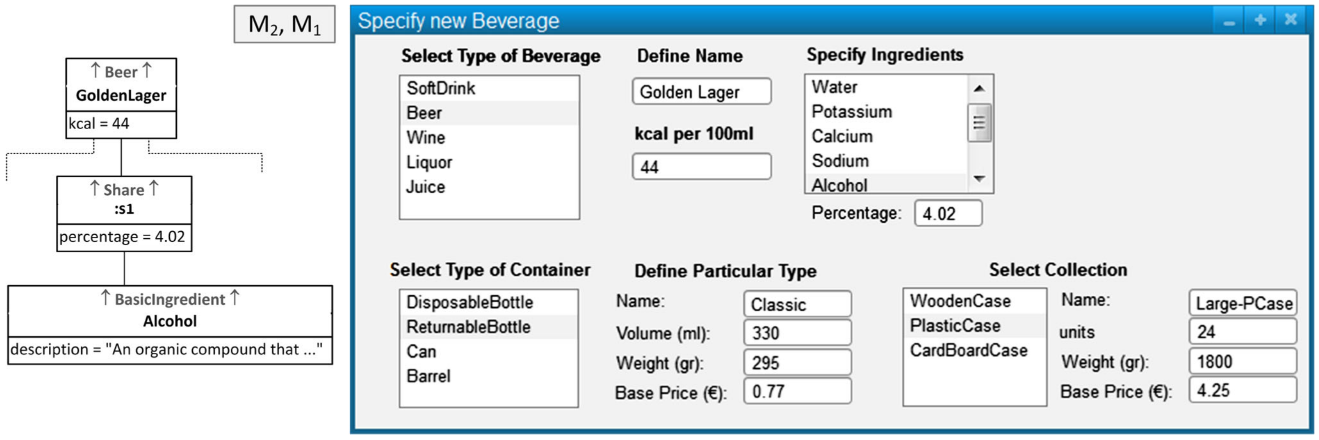


Fig. 11 DSML for modeling beverage types



Select Type of Beverage	Define Name	Specify Ingredients
SoftDrink, Beer, Wine, Liquor, Juice	Golden Lager	Water, Potassium, Calcium, Sodium, Alcohol
	kcal per 100ml: 44	Percentage: 4.02

Select Type of Container: DisposableBottle, ReturnableBottle, Can, Barrel

Define Particular Type

Name:	Classic
Volume (ml):	330
Weight (gr):	295
Base Price (€):	0.77

Select Collection: WoodenCase, PlasticCase, CardBoardCase

Name:	Large-PCase
units	24
Weight (gr):	1800
Base Price (€):	4.25

Fig. 12 Mockup of tool to specify classes of beverages and excerpt of corresponding diagram

6.1 Discussion

The main purpose of the proposed approach is to diminish the conflict inherent in designing DSMLs. With respect to this objective, a set of requirements were derived for the language architecture, a corresponding tool environment, and preserving coherence with traditional approaches to conceptual modeling. The evaluation of the approach was conducted from two perspectives (see **Table** 1). First, it was compared against the requirements (see the rows marked P1). Second, we evaluated

Table 1 Evaluation against requirements and comparison with traditional approaches

RLA-1: *Flexible number of classification levels*		
P1	The recursive language architecture enables an arbitrary number of classification levels.	+
P2	Traditional architectures such as MOF do not allow for an arbitrary number of classification levels.	−
RLA-2: *Relaxing the rigid instantiation/specialization dichotomy*		
P1	Intrinsic features allow the combining of aspects of specialization with aspects of instantiation.	+
P2	Similar concepts have been defined for traditional language architectures, but usually lack a respective implementation.	o
RLA-3: *No strict separation of language levels*		
P1	The FMML^x allows the creation of (meta) models that include classes on different classification levels.	+
P2	It is characteristic of traditional architectures that all classes within one model are on the same classification level.	−
RI-1: *Straightforward representation of language architecture*		
P1	The common representation of models and code is a nearly perfect approach to satisfy this demand.	+
P2	Traditional (meta) modeling tools are limited by programming languages that allow for one classification level only. Therefore, more levels can be represented only by overloading the M_0 level, which not only results in a conceptual mismatch but demands substantial implementation effort.	−
RI-2: *Cross-level integrity*		
P1	Since all classes of an entire language hierarchy are represented in a tool, changes on a higher level can be immediately implemented. Adding properties can be handled by automated updates of affected classes on lower levels. The deletion of properties requires manual intervention and is aggravated by the lack of static typing.	o
P2	In traditional tools, each editor operates on a particular level of classification. Models on different levels are not integrated. Therefore, it is far more demanding to support cross-level integrity.	−
RI-3: *Cross-level navigation*		
P1	All classes on all levels of classification are objects within one namespace. Therefore, navigation in any direction is not a problem.	+
P2	Since models on different levels of classification are usually not integrated in a tool, cross-level navigation is not possible without extraordinary effort.	−
RB-1: *Clear specification of classification levels*		
P1	Each class on every classification level is an object that stores its classification level.	+
P2	The classification level of a class can be determined from the model it is part of.	+
RB-2: *Backward compatibility*		
P1	The FMML^x includes concepts features by traditional meta-modeling languages as a subset.	+
P2	Not applicable.	

whether a traditional approach would be able to satisfy the requirements (see the rows marked P2). On the one hand, the term *traditional* refers to language architectures with a fixed number of classification levels. On the other hand, it refers to implementation languages that are restricted to one classification level. The symbols in the right-hand column indicate how well a respective requirement is satisfied according to the given justifications ("+": clearly satisfied, "o": partly satisfied, "−": not satisfied).

In addition to mitigating the DSML design problem, multilevel modeling promotes reuse and integration in general. Reuse of software artifacts across a range of applications requires those applications to share commonalities. If they lack common concepts on the M_1 level, direct reuse is not possible within current software architectures. Using multilevel language architectures for building information systems would enable reuse on higher levels of abstraction: Two systems that do not share common classes may well share common metaclasses or meta metaclasses. For the same reason, multilevel modeling fosters integration of software systems. The integration of two software systems requires them to share concepts in a common semantic reference system (Frank 2008), such as a common schema or a common set of classes. For instance, if two systems share the same product concept, they can efficiently exchange corresponding data; otherwise, integration would be compromised by the need to reconstruct semantics. If the two systems require specific product concepts, a high level of integration would not be possible. However, if the specific product concepts are based on a common, more generic concept, both systems could refer to this common concept and make sense of a corresponding instance. This would also enable more meaningful and efficient retrieval. Today, searching for product types depends on analyzing strings that might represent product names. Within a multilevel architecture, a product class would be an instance of a more general product (meta) class, which in turn might be instantiated from an even more general product class. All systems that are at least integrated via a reference DSML would be able to search for instances of generic product classes, including the instances of these instances.

The benefits of multilevel modeling are balanced by a few drawbacks. The flexibility enabled by a recursive language architecture is essentially based on replacing specialization with inheritance. This makes it possible for class A on M_1 to inherit from class B on M_2. It is even possible – and required for building classes on higher levels of classification – that a class on M_n inherits from a class on M_m, with $m < n$. As a consequence, the substitutability constraint (Liskov and Wing 1994), which is particularly useful for promoting consistent reuse, has to be sacrificed: An instance of a class on M_i could not replace an instance on M_j with $j <> i$ without generating a contradiction. Without the substitutability constraint, inheritance can be used as an instrument to enable selective reuse: If a certain operation of class A is also needed in class B, one could simply have B inherit from A regardless of whether any other feature of A is relevant for A. As a consequence, a class may inherit features it should not have, which creates a serious risk to system integrity. XMF leaves it to the developer to deal with this challenge. To reduce the risk of inheriting features that jeopardize integrity, we added the attribute *isCore: Boolean* to the XCore classes *Attribute* and *CompliedOperation* (see **Fig. 4**). Doing so allowed us to mark those features that should be inherited. On the level of subclasses, this information can be used to filter the list of all inherited features down to a useful selection by fading out those that are not marked with as *isCore*. Nevertheless, to avoid the risk created by the extensive use of multiple, cross-layer inheritance within XMF, developers would have to be familiar with the peculiarities of the language. A substantial amount of the flexibility that is enabled by XMF derives from the fact that it does not feature static typing, which may amongst others cause problems when checking the effects of deleting parts of a model. The lack of static typing can be compensated to some extent by analysis tools and by using pre- and post-conditions, which can be implemented in XMF with a moderate amount of effort.

6.2 Related Work

To the best of our knowledge, there is no approach that corresponds directly to the proposed conception of multilevel modeling. There are, however, various approaches that deal with the peculiarities of creating and using models on multiple classification levels. A few authors focus on investigating fundamental characteristics (Atkinson and Kühne 2001; Kühne 2006) and clarifying the respective terminology (Henderson-Sellers 2011). Others aim to dissolving the dichotomy between specialization and instantiation in order to reduce the complexity of multilevel model hierarchies. Such approaches include "Materialization" (Dahchour et al. 2002), "*m*-objects" (Neumayr et al. 2009), and the already mentioned powertypes (Odell 1994) and clabjects (Atkinson and Kühne 2008). While each approach has specific characteristics, they are all similar to that of intrinsic features. However, none of them is focused on the development of multilevel DSMLs. Instead, they are primarily intended to support systems development. While some of the approaches are supplemented by corresponding implementations (Kühne and Schreiber 2007; Atkinson et al. 2009), none of those makes use of a multilevel programming language. Volz presents an elaborate conceptual foundation for meta-modeling environments as well as a corresponding prototypical implementation that allows for multiple classification levels (Volz 2011). The repository is implemented in Java. Therefore, the semantics of multilevel instantiation is not embedded in the implementation language, as in XMF, but is based on an additional interpretation. Unlike the proposed conception of multilevel modeling, all those approaches are based on a traditional, MOF-style language architecture that is restricted to a certain number of classification levels, usually three. This restriction does not apply to "ConceptBase" (Jeusfeld 2009; Jarke et al. 1995), which allows an arbitrary number of classification levels. However, since it is implemented in Telos (Mylopolous et al. 1990), a declarative language based on predicate logic, it is different from our approach in various aspects. First, ConceptBase allows for deduction, which is not the case for XMF. Second, related to the first aspect, it uses a different concept of a class. As a consequence, the integration of ConceptBase with object-oriented programming languages has to overcome a serious semantic mismatch. ConceptBase is also not aimed at the development of DSMLs and corresponding tools.

Völter presents the idea of a "domain hierarchy," where "higher domains are a subset (in terms of scope) of the lower domains" (Völter 2013, p. 60). Apart from the fact that Völter uses an idiosyncratic terminology, where "higher" corresponds to "more specific", he does not detail how to develop and maintain hierarchies of languages. Kleppe outlines a vision of future domain-specific language systems where she distinguishes between *vernacular languages* and *vehicular languages*. Vernacular languages serve to cover a wide range of use scenarios, whereas vehicular languages are more specific, local languages that satisfy the needs of particular organizations (Kleppe 2009, preface). While Kleppe's vision clearly corresponds to the multilevel modeling approach presented, it is not further elaborated. Krogstie outlines a vision of empowering users of large enterprise systems through enterprise models, which not only promote a better understanding of complex domains but also enable advanced users to modify a system according to their needs. To increase the value of enterprise models, Krogstie demands that they be interactive (Krogstie 2007, p. 306). Unlike our work, Krogstie's considerations remain on an abstract conceptual level without consideration of implementation issues.

In the field of knowledge representation, also known as semantic web, there are a few languages based on description logic that permit the representation of various levels of classification. OWL Full (W3C 2004, 2009) is one of the most prominent representatives of these languages. However, although they provide powerful modeling concepts and support reasoning, they are not suited for our purpose, which is illustrated in the following by referring to OWL. First, OWL Full does not allow expression of the classification level of a class, which is a clear violation of requirement RB-I. Furthermore, the semantics of description logic, especially the conception of classes, is clearly different from those of object-oriented languages: Unlike description logic, an object is an instance of one and only one class (Frank 2012a). This kind of mismatch poses a serious challenge to the construction of tools based on (meta) models. Finally, OWL Full does not allow expression of whether an attribute of a metaclass is meant to represent a class or an instance-level feature. Hence, it would not be possible to represent intrinsic features. Walter et al. (2014) propose an integration of object-oriented metamodels and OWL metamodels. However, the scope is restricted to the MOF, hence to three classification levels.

7 Conclusions and Future Work

By reconstructing domain-specific concepts, DSMLs promise to substantially promote the economics of designing and using conceptual models and, hence, the economics of developing, managing, and maintaining information systems. However, their design is confronted with the fundamental conflict between range of reuse and productivity of reuse or, in other words, the ambivalent effects of semantics.

The approach presented in this paper clearly makes it possible to diminish this conflict. Furthermore, multilevel modeling enables relaxing the rigid dichotomy between instantiation and specialization, which contributes to reduced model complexity and allows for a more natural style of modeling since it corresponds directly to proven abstractions of natural languages. The more specific a DSML, the easier it is to use, because it provides clearer guidance and leaves less leeway for inappropriate constructions. Therefore, multilevel modeling fosters the empowerment of users by providing them with concepts they are familiar with in order to model and eventually change the domains and information systems for which they are responsible.

The proposed approach features the common representation of code and models. Therefore, it not only facilitates modeling tools that allow navigation of multiple language levels and that promote cross-level model integrity. Further, it enables executable models that allow users to directly query, analyze, and change models on various levels of abstraction. Lastly, it enables enterprise models to be integrated with enterprise software systems, thus providing the foundation of more advanced, self-referential enterprise systems (Frank and Strecker 2009). That in turn, could cause models to become the primary interface through which users could conceptualize, analyze, and modify enterprise software systems and their surrounding action systems.

Multilevel modeling is also suited to effectively promote integration. On the highest level, a generic common language could serve as a minimum standard to enable modest integration of all kinds of systems. Reference DSMLs for certain domains would provide common concepts on a higher level of semantics (i.e. with less leeway for interpretation) than generic concepts, thereby enabling a higher level of integration for all systems built with the reference DSML or one of the DSMLs that were created using the reference DSML. More specific DSMLs would enable tighter integration within narrower domains. Therefore, multilevel modeling is suited to reinvigorate the long-standing discussion on reference models: Instead of building one reference model for a certain domain, it is now possible to construct a hierarchical system of DSMLs that may include reference models on a lower level. So far, our work is restricted to static and functional abstractions. Preliminary investigations of applying multilevel modeling to dynamic abstractions such as process models are promising. However, the development of respective multilevel languages for process modeling is still a substantial challenge.

The specific advantages of the proposed approach to multilevel modeling mainly derive from a recursive language architecture that is in such clear contrast to MOF-like architectures that one can speak of a paradigm shift. It requires rethinking familiar concepts, especially since certain aspects of the new paradigm seem to be counterintuitive or even paradoxical, such as the possibility of inheriting from a class on a different classification level. At the same time, mastering high-level models such as those shown in **Fig. 7** or **Fig. 11** is not a trivial undertaking. Therefore, using the proposed approach appropriately requires experts willing to invest considerable time in understanding its central concepts. Nevertheless, even though higher-level models will often be complex, they are likely to be more comprehensible than the code found in some of today's enterprise software systems. Despite its use in various industrial projects, XMF is still a language that has not been widely disseminated. While this may be regarded as a serious disadvantage by IT managers, we do not consider it a substantial drawback from an academic perspective. Only if research takes the freedom to occasionally work with languages and tools that clearly deviate from mainstream solutions can it develop alternative and ultimately superior solutions that may foster progress in practice.

Further research is necessary to exploit the potential of multilevel modeling. First, there is a need to design and evaluate further hierarchies of DSMLs. Our preliminary investigations in the area of enterprise modeling indicate that resources, including IT infrastructures,

Abstract

Ulrich Frank

Multilevel Modeling

Toward a New Paradigm of Conceptual Modeling and Information Systems Design

Domain-specific modeling languages (DSMLs) promise clear advantages over general-purpose modeling languages. However, their design poses a fundamental challenge. While economies of scale advocate the development of DSMLs that can be used in a wide range of cases, modeling productivity demands more specific language concepts tuned to individual requirements. Inspired by the actual use of technical languages (German: "Fachsprachen"), this paper presents a novel multilevel modeling approach to conceptual modeling and to the design of information systems. Unlike traditional language architectures such as Meta Object Facility (MOF), it features a recursive architecture that allows for an arbitrary number of classification levels and, hence, for the design of hierarchies of DSMLs ranging from reference DSMLs to "local" DSMLs. It can not only diminish the conflict inherent in designing DSMLs, but enables the reuse and integration of software artifacts in general. It also helps reduce modeling complexity by relaxing the rigid dichotomy between specialization and instantiation. Furthermore, it integrates a meta-modeling language with a metamodel of a reflective meta-programming language, thereby allowing for executable models. The specification of the language architecture is supplemented by the description of use scenarios that illustrate the potential of multilevel modeling and a critical discussion of its peculiarities.

Keywords: Conceptual modeling, Information systems design, Meta-modeling, DSML, Executable models, Modeling tools, Modeling economics, Semantics

as well as goals, are promising subjects. Multilevel modeling may also be suited to promoting reuse in business process modeling, since current process modeling languages suffer from a substantial lack of abstraction (Frank 2012b). Second, the design and use of hierarchies of DSMLs present a number of specific challenges. From an economic point of view, there is a need to analyze whether the effort of creating a further level of abstraction can be justified by corresponding benefits such as improved economies of scale. From an epistemological point of view, the question is how to determine the appropriate number of classification levels for a certain domain. To analyze this question empirical investigations aiming to discover commonalities and differences within the domain of interest may seem appropriate. However, empirical studies alone will not be sufficient because creating DSMLs for reuse is not only based on reconstructing actual uses of technical languages. Instead, it will usually include a prescriptive element that is aimed at developing concepts better suited for certain purposes. Furthermore, guidelines are needed for organizing the design and maintenance of multilevel DSMLs. Since decisions may involve resolving the conflicting interests of language developers and users, coordination mechanisms are required to account for political aspects, as well. Finally, these decisions go beyond the scope of a certain organization or a particular industry. To take full advantage of multilevel DSMLs, interested parties would have to agree on hierarchies of DSMLs that cover many domains on a global scale.

We regard multilevel modeling as an very promising approach not only to move forward the field of conceptual modeling, but also for building, using and maintaining advanced information systems. In a joint project with one of the creators of XMF and the Xmodeler we further develop the Xmodeler, reconstruct an existing set of DSMLs for enterprise modeling and build a prototype of a self-referential enterprise system.

References

Atkinson C, Gutheil M, Kennel B (2009) A flexible infrastructure for multilevel language engineering. IEEE Transactions on Software Engineering 35(6):742–755

Atkinson C, Kühne T (2001) The essence of multilevel metamodeling. In: Gorgolla M, Kobryn C (eds) UML 2001 – the unified modeling language: modeling languages, concepts, and tools. Proc of the 4th international conference, Toronto, Canada, October 1–5, 2001. Springer, Berlin, pp 19–33

Atkinson C, Kühne T (2008) Reducing accidental complexity in domain models. Software & Systems Modeling 7(3):345–359

Clark T, Sammut P, Willans J (2008a) Applied metamodelling: a foundation for language driven development. 2nd edn. Ceteva. https://eprints.mdx.ac.uk/6060/1/Clark-Applied_Metamodelling_%28Second_Edition%29%5B1%5D.pdf. Accessed 2014-10-21

Clark T, Sammut P, Willans J (2008b) Superlanguages: developing languages and applications with XMF. Ceteva. https://eprints.mdx.ac.uk/6079/1/Clark-Superlanguages%5B1%5D.pdf. Accessed 2014-10-21

Clark T, Sammut P, Willans J (2008c). Applied metamodelling: a foundation for language driven development

Dahchour M, Pirotte A, Zimanyi E (2002) Materialization and its metaclass implementation. IEEE Transactions on Knowledge and Data Engineering 14(5):1078–1094

Fettke P, Loos P (eds) (2007) Reference modeling for business systems analysis. Idea Group, Hershey

Fowler M (2011) Domain-specific languages. Addison-Wesley, Upper Saddle River

Fill H, Karagiannis D (2013) On the conceptualisation of modelling methods using the ADOxx meta modelling platform. Enterprise Modeling and Information Systems Architectures 8(1):4–25

Frank U (2002) Modeling products for versatile e-commerce platforms essential requirements and generic design alternatives. In: Arisawa H, Kambayashi Y, Kumar V, Mayr HC, Hunt I (eds) Conceptual modeling for new information system technologies. Springer, Berlin, pp 444–456

Frank U (2008) Integration – reflections on a pivotal concept for designing and evaluating information systems. In: Kaschek R, Kop C, Steinberger C, Fliedl G (eds) Information systems and e-business technologies. Proc 2nd International United Information Systems Conference UNISCON 2008, Klagenfurt, Austria, April 22–25, 2008. Springer, Berlin, pp 11–22.

Frank U (2011a) The MEMO meta modelling language (MML) and language architecture. 2nd edn. ICB-research report, Institute for Computer Science and Business Information Systems, University Duisburg-Essen, No 43

Frank U (2011b) Multi-perspective enterprise modelling: background and terminological foundation. ICB-research report, Institute for Computer Science and Business Information Systems, University Duisburg-Essen, No 46

Frank U (2012a) Thoughts on classification/instantiation and generalisation/specialisation. ICB-research report, Institute for Computer Science and Business Information Systems, University Duisburg-Essen, No 53

Frank U (2012b) Specialisation in business process modelling: motivation, approaches and limitations. ICB-research report, Institute for Computer Science and Business Information Systems, University Duisburg-Essen, No 51

Frank U, Strecker S (2009) Beyond ERP systems: an outline of self-referential enterprise systems: requirements, conceptual foundation and design options. ICB-research report, Institute for Computer Science and Business Information Systems, University Duisburg-Essen, No 31

Henderson-Sellers B (2011) Random thoughts on multi-level conceptual modelling. In: Kaschek R, Delcambre L (eds) The evolution of conceptual modeling: from a historical perspective towards the future of conceptual modeling. Springer, Berlin, pp 93–116

Hofstadter DR (1979) Godel, Escher, Bach: an eternal golden braid. Basic Books, New York

Jarke M, Eherer S, Gallersdörfer R, Jeusfeld M, Staudt M (1995) Concept base – a deductive object base for meta data management. Journal of Intelligent Information Systems 4(2):167–192

Jeusfeld MA (2009) Metamodeling and method engineering with ConceptBase. In: Jeusfeld MA, Jarke M, Mylopoulos J (eds) Metamodeling for method engineering. MIT Press, Cambridge, pp 89–168

Kelly S, Tolvanen J (2008) Domain-specific modeling: enabling full code generation. Wiley-Interscience/IEEE Computer Society, Hoboken

Kelly S, Lyytinen K, Rossi M (2013) MetaEdit+ a fully configurable multi-user and multi-tool CASE and CAME environment. In: Bubenko J (ed) Seminal contributions to information systems engineering: 25 years of CAiSE. Springer, Berlin, pp 109–129

Kleppe AG (2009) Software language engineering: creating domain-specific languages using metamodels. Addison-Wesley, Upper Saddle River

Krogstie J (2007) Modeling of the people, by the people, for the people. In: Krogstie J, Opdahl A, Brinkkemper S (eds) Conceptual modelling in information systems engineering. Springer, Berlin, pp 305–318

Kühne T (2006) Matters of (meta-)modeling. Software & Systems Modeling 5(4):369–385

Kühne T, Schreiber D (2007) Can programming be liberated from the two-level style: multi-level programming with deepjava. In: Gabriel RP, Bacon DF, Lopes CV, Steele GL (eds) Proc of the 22nd annual ACM SIGPLAN conference on object-oriented programming systems and applications (OOPSLA '07). ACM Press, New York, pp 229–244

Liskov BH, Wing JM (1994) A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems 16:1811–1841

Mahr B (2009) Die Informatik und die Logik der Modelle. Informatik-Spektrum 32(3): 228–249

Morin B, Barais O, Jézéquel J, Fleurey F, Solberg A (2009) Models@Run: time to support dynamic adaptation. IEEE Computer 42(10):46–53

Mylopolous J, Borgida A, Jarke M, Koubarakis M (1990) Telos: representing knowledge about information systems. ACM Transactions on Information Systems 8(4):325–362

Neumayr B, Grün K, Schrefl M (2009) Multi-level domain modeling with m-objects and m-relationships. In: Kirchberg M, Link S (eds) Conceptual modelling 2009: proc of the 6th Asia-Pacific conference on conceptual modelling (APCCM 2009). Australian Computer Society, Sydney, pp 107–116

Object Management Group (2006) Meta object facility (MOF) core specification: version 2.0

Odell JJ (1994) Power types. Journal of Object-Oriented Programming 7(2):8–12

Schütte R (1998) Grundsätze ordnungsmäßiger Referenzmodellierung: Konstruktion konfigurations- und anpassungsorientierter Modelle. Gabler, Wiesbaden

Völter M (2013) DSL engineering: designing, implementing and using domain-specific languages. dslbooks.org

Volz BW (2011) Werkzeugunterstützung für methodenneutrale Metamodellierung. Dissertation, University of Bayreuth

Walter T, Parreiras FS, Staab S (2014) An ontology-based framework for domain-specific modeling. Software & Systems Modeling 13(1):83–108

W3C (2004) OWL web ontology language: W3C recommendation 10 February 2004. http://www.w3.org/TR/owl-ref/. Accessed 2014-10-21

W3C (2009) OWL 2 web ontology language: W3C recommendation 27 October 2009. http://www.w3.org/2009/pdf/REC-owl2-overview-20091027.pdf. Accessed 2014-10-21