Toward a unified conception of multi-level modelling: advanced requirements

Ulrich Frank

University of Duisburg-Essen, Germany
ulrich.frank@uni-due.de

Abstract. Apart from sharing common concepts, current approaches to multi-level modelling differ with respect to specific features and the terminology. Both are an obstacle for the further development of the field. This paper presents a selection of possible requirements for advanced multi-level modelling languages. It is intended as a contribution to a broader discussion of requirements and to the development of a common research agenda.

1 Introduction

Various languages and tools have been proposed to support multi-level modelling [1], [2], [3], [4], [5]. Apart from differences regarding specific aspects, they all share a few essential properties. First, they allow for an arbitrary number of classification levels. Second, every class, no matter on what level, is an object at the same time. Third, they allow for deferred or deep instantiation, which means that attributes of a class do not have to be instantiated with the direct instances of that class, but only later in instances of instances. Furthermore, in most cases, approaches to multi-level modelling support "strict meta-modelling" [6], that is, every object in a multi-level model is assigned a classification level. The lack of a unified or even standardized language for meta-modelling is an obstacle to the further dissemination of multi-level modelling. At the same time, it fosters the competition between different approaches, which is better suited to advance the field than a standard that is likely to freeze a certain state. Nevertheless, competition creates a problem, if it hinders collaboration. The development of a language for multi-level modelling, and even more so, the implementation of corresponding tools requires a substantial effort. Therefore, it seems natural that the developers of a certain approach favour their concepts and tend to protect them against others. Against this background, it seems more promising to start the competition not only with the presentation of language specifications, but during requirements analysis already. First, the more researchers are involved in the analysis of requirements, the more likely it is to identify relevant issues. Second, from a psychological perspective, it seems to be easier to develop a consensus on requirements than on the comparative evaluation of existing artefacts – at least as long as the creators of these artefacts are involved.

This paper is intended to serve as a contribution to starting a community initiative on discussing requirements for future, possibly unified multi-level modelling languages that go beyond the common concepts outlined above. The requirements resulted from analysing limitations experienced with the use of the multi-level modelling language FMML^x [7], as well as from discussions during various workshops and meetings. The requirements are illustrated in part with diagrams that were created with the FMML^x . Fig. 1 gives an overview of essential elements of the concrete syntax. Colors are used to represent classification levels.

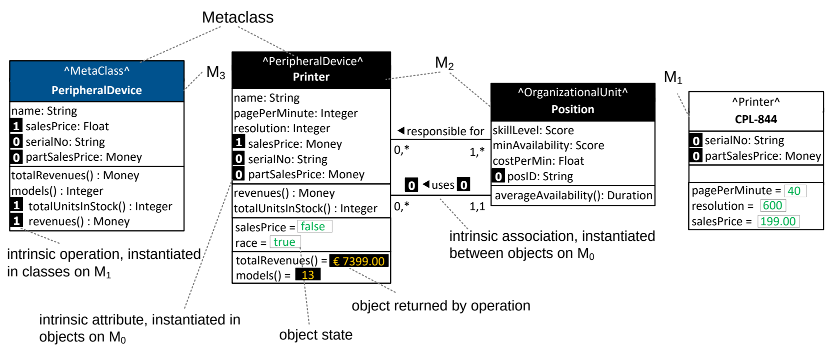

Fig. 1. Illustration of the notation of the FMML^x

2 Requirements

The proposed requirements are restricted to those that belong to the core of a language. Additional requirements that concern the size of a language, or the management and analysis of models are not accounted for. To stay within the space limitations, various requirements had to be omitted. Some of those will be mentioned in the conclusions.

2.1 Contingent levels

Every class specified in the FMML^x has to be assigned an explicit classification level. There is a good reason for this constraint, sometimes referred to as “strict multi-level modelling” [6, p. 28]. With respect to the semantics of a class, it makes a clear difference whether it is meant as a class on M_1 or on any M_n with $n > 1$. 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 modelling it is preferable to avoid ambiguity. Nevertheless, there are cases where the need to assign a strict, invariant level to a class is problematic, because it prevents an abstraction that would be useful, e.g., to increase reusability. The appropriate number of classification levels does not only depend on the variety of the subject, but also on the need to express variety. Sometimes, the design of two classification hierarchies of similar objects results in a top level meta class that makes sense for both hierarchies, but that would be located in one hierarchy on a level different from the one needed in the other hierarchy. The example in fig. 2 illustrates this problem. In the example, the class `Product` on M_4 is instantiated into `PeripheralDevice` on M_3 and into `Desk` on M_2. According to the experience gathered during the last years this is not an exotic exception, but occurs regularly.

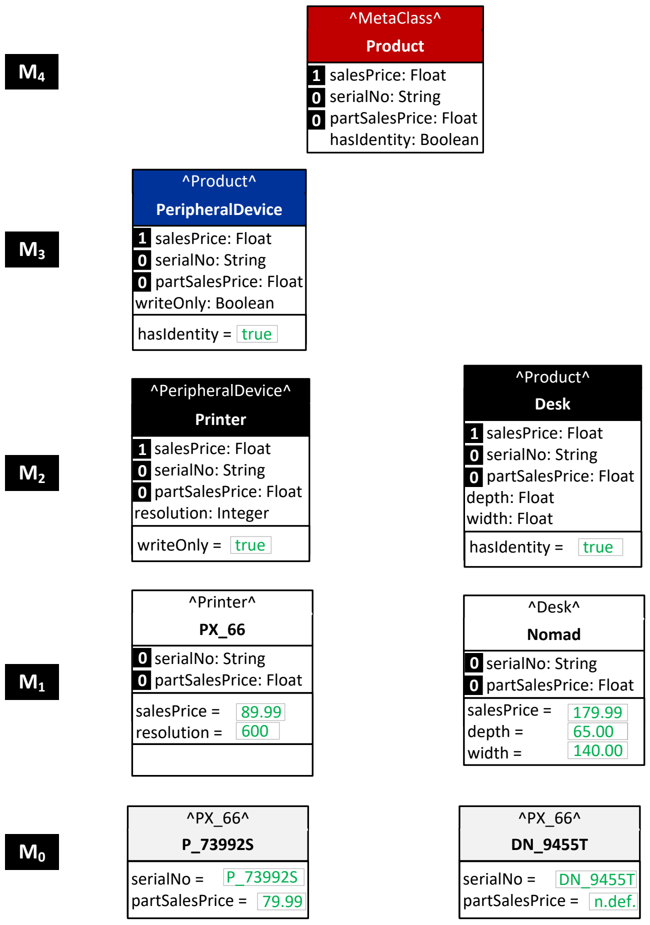

Fig. 2. Illustration of problem produced by strict levels

R1 In addition to having classes with a definite level, it should be possible to define classes with a *contingent* level. A contingent level allows for adapting the concrete level of a class to different contexts of use. *Rationale*: It happens that two classes on different classification levels could be classified by the same meta class. However, this can be achieved only, if the meta class does not have a definite classification level. Instead, its level would have to be contingent with respect to the instantiation context.

Corresponding solutions are proposed by [8] who suggest to allow for "leaping" levels, and by [9] who add flexibility by replacing numbers to qualify levels with labels.

Priority: *high*

Challenge: The definition of a class as level-contingent has a substantial impact on a model's integrity. It is not necessarily the case that an attribute defined in a contingent class can be instantiated on any level. If, for example, an attribute is intended to store the average customer satisfaction with a certain bicycle model or bicycle type, it would make sense only to instantiate om M_1 or above.

Challenge: The requirement demands for a cross-level constraint language.

2.2 "Classless" classes

Like with any object-oriented models, the design of multi-level models requires the creation of classes. In traditional, one-level models, a class is implicitly instantiated from one specific meta-class. In multi-level models, a new class is instantiated from a class that exists in the model already. As a consequence, the design of a multi-level model requires a top-down approach, that is, one has to start with classes on the topmost classification level. Then, classes on lower levels have to be added step by step. There are cases where it may be perceived as inappropriate to be forced to a top-down approach. Sometimes, the topmost classes are not known in advance. Instead, they may result through an act of abstraction from lower level classes. In other words, it would be helpful, if a top-down approach was supplemented by a bottom-up approach. That would require allowing for the preliminary creation of classes without an explicit meta-class.

R2 It should be possible to define preliminary classes without a meta-class. *Rationale*: Sometimes, a bottom-up approach seems to be more appropriate than a top-down approach.

Priority: *high*

Challenge: From a technical perspective, it is not possible that a class does not have a meta-class. Therefore, some preliminary meta-class has to be assigned, which would be later replaced by the final meta-class. Hence, a corresponding modelling tool needs to support meta-class migration.

2.3 Distinction of instantiation levels within associations

Like attributes and operations, associations could be defined as intrinsic, too. The example in fig. 3 illustrates the idea. The classes `BusinessProcess` and `Position` are both located on M_2, that is, they are supposed to be instantiated into process types and position types. The intrinsic association between both is to express that a particular process instance on M_0 is assigned to a particular instance of a position type, also on M_0. This is indicated by the zero printed in white on the black square next to the association name. It is not a rare exception that an association should be instantiated on different levels with both associated classes. The example in fig. 3 includes the association "responsible for". It is to express that a particular position instance on M_0 is responsible for a type of platform on M_1.

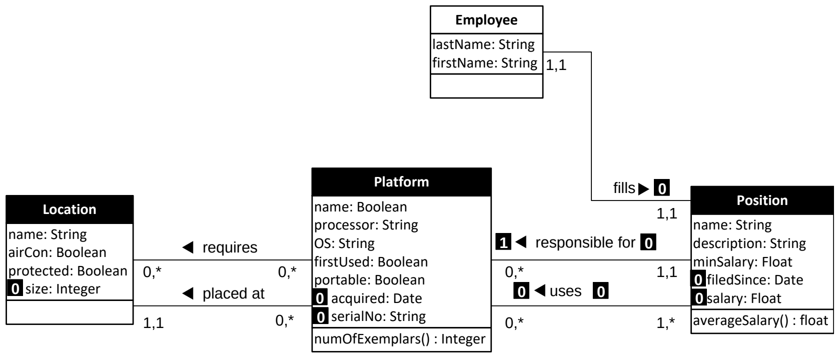

Fig. 3. Different instantiation levels of an intrinsic association

R3 Intrinsic associations should allow the specification of different instantiation levels for both participating classes. *Rationale*: It is common in natural language to link two concepts (or object references) on different classification levels in one sentence. Accordingly, it can be a relevant requirement to link two objects on different classification levels in an multi-level model. If the association can be defined on a higher level already, that is, if the association is intrinsic, it follows directly that it must be possible to support different instantiation levels.

This feature is already part of the current implementation of the FMML^x in the Xmodeler and proved to be very useful.

Priority: *high*

2.4 Avoiding redundant specification

It happens that classes within a classification hierarchy share the same properties, that is, the same attributes, operations, or associations. A frequent example is an attribute like "name", which may be required for a class, its instances and all further instances down the instantiation line. Further examples comprise operations that perform statistics on object populations. It might be required for every class within a classification hierarchy to provide methods that calculate the number of direct instances or the cumulated number of all instances of instances of a class (see example in fig. 4).

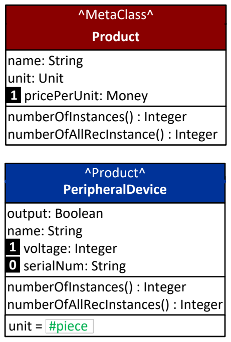

Fig. 4. Example of repeated specification of the same properties

R4 It should be possible to indicate that certain properties of a class are to be inherited to its instances. *Rationale*: Inheriting features where it is appropriate enables preventing redundancy.

Note that inheriting properties is different from intrinsic properties. An intrinsic property is not directly instantiated where it is defined, but only at the specified instantiation level. An inheritied property can be part of many classes. Therefore, it can be instantiated multiple times. Intrinsic properties must not be explicitly inherited, since explicit inheritance of intrinsic properties would be redundant and would therefore compromise the comprehensibility of a model. In an ideal case, the implementation of operations can be inherited, too (as it would be the case with the example in fig. 4).

Priority: *high*

In the current implementation of the FMML^x , this problem is relaxed implicitly. Certain properties, such as the attribute `name`, are defined with the central class `Class` in Xcore. Every class that is defined with the FMML^x inherits through `MetaClass` from `Class`. However, this approach works only for properties that are generic enough to be inherited to all classes of an entire system.

2.5 Support for the specification of association types

Associations are of pivotal relevance for the design of languages and models. Current languages do not provide specific support for the specification of association types. If an association is characterized by specific semantics, it has to be expressed by additional constraints. Table 1 gives an overview of general association types.

name	*description*	*semantics*	*variants*	*behaviour*
interaction	no specific semantics	none, restricted to cardinalities	state dependency, *e.g.* marriage, sex	transparent creation of links
aggregation	particular objects that are part of another object, *e.g.* a wheel that is part of a particular car	special case of interaction	state dependency; dependent on corresponding composition	transparent creation of links
composition	an abstraction over aggregation, *e.g.* the construction of a car model may allow for 1 .. * wheel types	similar to aggregation, however, on a different level of abstraction (a type is an aggregation of other types)	state dependency, *e.g.* any type of wheel that is certified for a certain speed	transparent creation of links including ownership, requirements for MLM
delegation	transparently access data or behaviour of another object	special case of interaction; *delegator* delegates behaviour (and, hence, state) to *delegatee*	state dependency, *e.g.* age, qualification, authorization	transparent creation of links; object creation; message dispatch
delegation to class	transparently access data or behaviour that is common to all instances of a class, *e.g.* a price of a product exemplar that is specified with the corresponding class	similar to delegation, however on a different level of abstraction. *Delegatee* (object) delegates behaviour to its *class*	delegation may be restricted to classes that are marked as delegatees	transparent creation of links; object creation; message dispatch; requires rules to define dispatch order; MLM requirements

Table 1. Example association types

In addition to general association types, there are domain-specific association types. Their semantics depends on domain-specific requirements, which may vary. Hence, the language should offer meta-association types that allow the creation of various association types of the same kind. Also, the semantics of a domain-specific association type may depend on properties of the associated classes. Table 2 illustrates the idea of domain-specific association types with a few examples.

R5 The language should provide generic association types. At best, these types would be provided as instances of an association meta type. *Rationale*: Including generic association types promotes modelling productivity and model

name	*kinds of classes*	*description*	*variations*
married to	Person, Person	marriage depends on certain requirements, e.g. sex, age	different constraints on sex, multiplicities, age
holds	Person, Driving Licence	holding a driving licence requires people to satisfy certain requirements	constraints on min. and max. age, gender, different types of driving licences

Table 2. Examples of domain-specific association types

integrity. Meta association types provide modellers with more flexibility, since they allow to vary the semantics of generic association types.

R6 It should be possible to define domain-specific association types. In order to cover the variance of domain-specific association types, it could be helpful, if a language provides meta-association types on a higher level, that would allow the instantiation of meta-association types. *Rationale*: The specification of domain-specific association types promotes model integrity.

Priority: *medium*

Challenge: The specification of meta-associations is very demanding, because association types can hardly be defined on their own, that is, without regard to the classes they connect. For meta-associations, that would require adequate meta-classes, which, however, can hardly be predicted in advance.

2.6 Semantic enrichment of properties

Properties of a class in a multi-level system may serve different purposes. An attribute that is instantiated in a class may represent properties of that class, a property value shared by all instances of the class, or constraints on possible property values in instances. The example in fig. 5 illustrates this issue with respect to attributes. The attribute `minQualityLevel` is instantiated with all instances and serves the specification of a constraint on possible values of instances of these instances (all types of printers in that system). The attribute `serialNum`, though being an intrinsic attribute, is supposed to be instantiated in individual values for each instance (particular printers). Finally, attributes like `pagesPerMin` are to be instantiated into values that are shared by all instances of the class a particular value was assigned to.

Similar distinctions can be made for operations. The values delivered or used by corresponding access operations would represent corresponding aspects. Associations may also be used to create links between an object (e.g. representing a country) and a class (e.g. representing a specific product type), where the link is semantically shared by all its instances.

R7 A language should allow for clearly distinguishing between attributes that are regularly instantiated into individual values of instances, that are instantiated into values of instances that actually represent common values of their

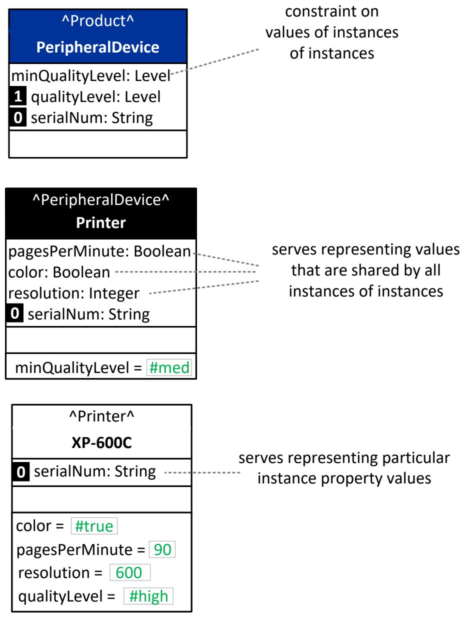

Fig. 5. Different kinds of attributes

instances, and those that serve the specification of constraints on attribute values. At the same time, corresponding access operations should be qualified accordingly. *Rationale*: These different kinds of properties have specific semantics that would be lost, if there was no way to express it somehow, which in turn could compromise the integrity of systems.

This requirement is fairly easy to satisfy and promises clear benefits.
Priority: *high*

There are also different kinds of operations. Generic categories of operations include *instantiation*, *read access*, *write access*, *update* or *release*.

R8 It should be possible to assign categories to operations. *Rationale*: Distinguishing categories of operations enables more meaningful model queries and fosters models integrity.

Priority: *high*

3 Conclusions

Despite its undisputed benefits, multi-level modelling has still not reached the mainstream of conceptual modelling and software engineering. Therefore, it would be important to bundle resources and to start a collaboration beyond existing approaches. That recommends not only focussing on "main open questions in multi-level modeling" [10, p. 54] including the appropriate size of a language, but also on requirements for future extensions of multi-level modelling languages. These activities are suited to foster the evolution of a common research agenda and a unified terminology – even though some of the requirements are already satisfied by existing approaches. The requirements presented in this paper are only a starting point. Various requirements had to be omitted because of space limitations. To name two examples only: it would be useful to allow for the specification of attributes with classes above M_1, and, more important, there is need for a multi-level object constraint language.

References

1. M. A. Jeusfeld, Metamodeling and method engineering with conceptbase, in: M. A. Jeusfeld, M. Jarke, J. Mylopoulos (Eds.), Metamodeling for Method Engineering, MIT Press, Cambridge, 2009, pp. 89–168.
2. T. Kühne, D. Schreiber, Can programming be liberated from the two-level style: multi-level programming with deepjava, in: R. P. Gabriel, D. F. Bacon, C. V. Lopes, G. L. Steele (Eds.), Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications (OOPSLA '07), Vol. 42,10 of ACM SIGPLAN notices, ACM Press, New York, 2007, pp. 229–244.
3. C. Atkinson, R. Gerbig, Flexible deep modeling with melanee, in: S. B. U. Reimer (Ed.), Modellierung 2016, 2.-4. März 2016, Karlsruhe - Workshopband, Vol. 255 of Modellierung 2016, Gesellschaft für Informatik, Bonn, 2016, pp. 117–122.
4. C. Atkinson, B. Kennel, B. Goß, The level-agnostic modeling language, in: B. Malloy, S. Staab, M. van den Brand (Eds.), Software Language Engineering, Vol. 6563 of LNCS, Springer Berlin Heidelberg, 2011, pp. 266–275.
5. J. de Lara, E. Guerra, Deep meta-modelling with metadepth, in: J. Vitek (Ed.), Objects, Models, Components, Patterns, 48th International Conference, TOOLS 2010, Málaga, Spain, June 28 - July 2, 2010. Proceedings, Vol. 6141 of Lecture Notes in Computer Science, Springer, 2010, pp. 1–20. doi:10.1007/978-3-642-13953-6.
6. C. Atkinson, T. Kühne, The essence of multilevel metamodeling, in: M. Gorgolla, C. Kobryn (Eds.), UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools, Vol. 2185 of Lecture Notes in Computer Science, Springer, Berlin and London, New York, 2001, pp. 19–33.
7. U. Frank, Multilevel modeling: Toward a new paradigm of conceptual modeling and information systems design, Business and Information Systems Engineering 6 (6) (2014) 319–337.
8. J. D. Lara, E. Guerra, R. Cobos, J. Moreno Llorena, Extending deep meta-modelling for practical model-driven engineering, The Computer Journal 57 (1) (2014) 36–58.
9. B. Neumayr, K. Grün, M. Schrefl, Multi-level domain modeling with m-objects and m-relationships, in: S. Link, M. Kirchberg (Eds.), Proceedings of the 6th Asia-Pacific Conference on Conceptual Modeling (APCCM), Australian Computer Society, Wellington, 2009, pp. 107–116.
10. C. Atkinson, R. Gerbig, T. Kühne, Comparing multi-level modeling approaches, in: MULTI 2014 : Proceedings of the Workshop on Multi-Level Modelling, Vol. 1286, RWTH, Aachen, 2014, pp. 53–61.