

# Transforming a large Flat Model into a Multi-Level Model. A Retrospective Case Study

Ulrich Frank  
Computer Science  
Universität Duisburg-Essen  
Essen, Germany  
ulrich.frank@uni-due.de

## Abstract

This paper presents a case study comprising the analysis and transformation of a large flat object model into a multi-level model. To support the transformation, a corresponding method is introduced and applied. The resulting multi-level model is evaluated against the original requirements as well as with regard to further aspects.

## CCS Concepts

• **Software and its engineering** → **Object oriented architectures; Domain specific languages.**

## Keywords

Multi-Level Modeling, Model Transformation, Trading Platform

## ACM Reference Format:

Ulrich Frank. 2026. Transforming a large Flat Model into a Multi-Level Model. A Retrospective Case Study. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026)*, October 04–09, 2026, Málaga, Spain. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3837062.3838701>

## 1 Introduction

The advantages of multi-level modeling (MLM) have long been recognized. They include, among other things, the promotion of reuse, integration, and adaptability. In a number of studies, these advantages are theoretically attributed to the undeniable gain in abstraction afforded by multi-level language architectures. Nevertheless, MLM remains largely unknown in practice. Case studies that involve models of a scale found in real-world projects should be suited to increasing awareness of MLM in practice. To this end, it is also necessary to make clear that a move toward MLM does not render investments in existing flat models and schemas obsolete, but rather allows for a gradual transformation. To my knowledge, corresponding case studies are not available. The present paper addresses this gap by presenting a case study that comprises a real world project of considerable size and its reconstruction as a multi-level model. Using this project for a case study is unusual, since the original project was carried out about 25 years ago. It was inspired by a startup that began at a university and in which I was involved.

---

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only. Request permissions from owner/author(s). *MODELS Companion 2026, Málaga, Spain*  
© 2026 Copyright held by the owner/author(s).  
ACM ISBN 979-8-4007-2903-4/26/10  
<https://doi.org/10.1145/3837062.3838701>

The case study investigates the effort required and the benefits gained from reconstructing large flat models into multi-level models. To this end, it also aims at evaluating a method specifically designed to support this reconstruction process.

The investigation of the case is structured as follows. First, the business model is briefly described. Next, the key requirements are discussed, with particular emphasis on the associated challenges for system design. This is followed by an account of how these challenges were addressed in the original solution. Against the background of the weaknesses of that solution, the data model is then reconstructed as a multi-level model. To this end, a methodical approach is adopted that provides for an extension of the original requirements and supports, among other things, the identification of suitable opportunities for abstraction. Finally, the proposed multi-level model is evaluated against the extended requirements.

## 2 The Case

The late 1990s were marked by an almost euphoric sense of new departure, triggered above all by the rapid spread of the World Wide Web. This atmosphere was accompanied by a considerable number of start-ups whose business models were, in one way or another, aimed at exploiting the commercial potential of the Internet. Early online marketplaces appeared particularly impressive, as they promised dramatic gains in efficiency made possible by a high degree of automation and considerable economies of scale. The information systems required to realize such trading platforms attracted our interest, particularly with regard to one specific aspect. The nature and scope of the products offered – or, more precisely, of the product types – are not fixed when designing such systems, but are subject to continuous change during system operation. In addition, it may be necessary to support customer-specific product configuration. The required database schema, as well as the data or object structure of the corresponding software, would therefore have to be sufficiently flexible to allow the necessary extensions to be carried out at runtime.

We had no insight into the database structure of platforms such as Amazon, but we had the impression that product types were modeled by means of a relatively simple, generic structure. Product-specific details appeared to be stored in supplementary texts or as structured character strings. Our initial interest, therefore, was in designing a schema that would permit the required extensions at runtime while at the same time enabling a high degree of consistency – something that could hardly be achieved by means of supplementary XML documents alone. After we had developed a first approach to address this challenge, we began by asking ourselves, hypothetically, whether we might make use of the prevailing

enthusiasm of the time by founding a company on that basis. This, however, required a business model that would allow for clear and sustainable differentiation from existing online marketplaces.

## 2.1 Business Model

The business model was based on the idea of offering unrivaled conditions for selected products. To this end, the focus was to be placed on products for which the gap between production costs and sales price is particularly large, and which therefore appeared to offer substantial potential for price reductions.

Three central measures were intended to substantially reduce prices for such products:

- highly automated processes, in order to keep the platform's operating costs low and to enable a high degree of scalability;
- aggregation of demand, in order to obtain favorable conditions. From the suppliers' perspective, such aggregation is associated with a loss of pricing power, but it also offers cost advantages by reducing planning uncertainty. This applies in particular where production takes place only after a bundled order has been secured;
- minimization of the risk of payment default. Demand aggregation can function only if all parties involved are contractually committed to accepting and paying for the quantity ordered.

The following example process illustrates the idea. A market analyst identifies a product that seems suited for demand aggregation. This might be a luxury sofa with a regular sales price of € 9,999.00. Negotiations with the manufacturers result in an offer of € 4,230.00 for an order of 200 and € 3,990.00 for an order of 400. By adding further costs, e.g., for distribution (requiring a contract with a logistics firm) and marketing, a *demand forum* is initiated together with a call for participation. It announces a price of € 4,499.00 per sofa in case that 200 buyers join in until a certain deadline – and a price of € 4,250.00 respectively, if at least 400 orders will be collected. Based on this, the forum transitions to the “activated” state (or will be closed in the case of failure). Participating in a forum requires the signing of a preliminary purchase agreement. In addition, a bank guarantee for the full purchase price is required. Once the required number of buyers has been reached, the order is placed with the manufacturer. Once the sofas are ready, buyers will be contacted to schedule a delivery date. Payment is due upon delivery. Except for the initial market analysis and the negotiations with the manufacturer the entire process, including the request for a bank guarantee, is fully automated.

## 2.2 Requirements and Challenges

The original requirements comprised two major subjects. The pursuit of a high degree of automation makes it necessary to design all processes so that they run largely automatically. The corresponding requirements, which are not in the primary scope of this paper, were captured by 10 business process models, each of which was supplemented by textual requirements assigned to subprocesses and events.

The other subject of requirements analysis was focussed on data management. The analysis covered the following subdomains:

- actors and roles (human/organizational, employee, customer, bank, manufacturer, logistic firm, service provider, etc)
- international addresses
- documents (order, invoice, complaint, etc)
- contracts (delivery contract, participation contract, broker contract, leasing contract, etc)
- payment (credit card, bank account, transaction, etc)
- products (physical products, financial products, product configurations, packaging, etc)

Requirements analysis resulted in a large number of specific requirements, which cannot be accounted for in their entirety here. Instead, I will focus on requirements that are of particular importance for the case study because they present significant challenges (see also [1]). They concern the representation of products and contracts.

- Req. 1 It should be possible to represent a wide range of substantially different product types. *Rationale:* This requirement is directly related to the idea of a platform that covers a variety of product types.
- Req. 2 It should be possible to assign an id and the production date to particular product instances. *Rationale:* Different from the order contract, the final confirmation of delivery may require referring to a particular product instance and its characteristics.
- Req. 3 Customers should be enabled to specify particular product configurations. *Rationale:* Many products that are addressed by the business model are offered in different variants. To minimize costs, configuration should be enabled by the system without the need for a sales person to provide support.
- Req. 4 Adding new product types or product configurations should be possible at runtime. *Rationale:* It would cause an unreasonable disruption to business operations if adding new product types required recompiling the system.
- Req. 5 Product descriptions should be detailed in order to automate communication with various external partners (banks, suppliers, etc.) as much as possible. *Rationale:* Automated communication with external partners is essential for reducing process costs.
- Req. 6 Users should be largely prevented from specifying invalid product types. *Rationale:* A specification that differs from the actual characteristics of the products in question can result in serious financial losses.
- Req. 7 The system should provide for various types of contracts and allow for the representation of individual contracts, too. *Rationale:* Contracts are essential for running the business model. Individual contracts include additional information such as the name of the parties that signed it and the date when they were signed.
- Req. 8 The model should allow for fine-grain queries that include specific details of products. *Rationale:* With a growing number of product types and products, support for finding the right product gets increasingly important.

The range of products to be managed by the system was not restricted to physical products, but comprised also financial products. Further requirements needed to be accounted for, though not being at the center of our attention. The representation of products posed

the greatest challenge, since the introduction of new product types was intended to be possible at runtime. By contrast, we assumed that the introduction of new contract or document types would be required only comparatively rarely, and that it would therefore be acceptable to recompile the system in each such case.

### 3 Conceptual Foundation of the Original System

When designing a viable solution, there was one technical constraint to consider. The integrity of the managed data was of paramount importance, as it formed the basis for binding contracts with various partners. For this reason, we opted for an object-relational database. In addition, we have chosen Java as the programming language. It quickly became clear that meeting the requirements listed above would involve some significant challenges. To meet Req. 1 and Req. 3, the database schema – and the class structure of the Java code – would actually need to be modified, but that would violate Req. 4. Given the wide variety of possible product types, Req. 6 raised significant doubts about the feasibility of a convincing solution.

#### 3.1 The Original Object model

By using a database development environment (Oracle Designer with Oracle 9i), we were able to generate the schema largely from the object model. The proprietary language used for the design of the object model was similar to UML. However, the effort required to create a comprehensive object model and the corresponding Java programs was considerable. We were therefore forced to relax some of the requirements. This concerned mainly Req. 3 and Req. 6. Req. 3 was addressed using a pragmatic approach, while we, at first, set aside Req. 6.

The resulting object-relational object model comprised 100 classes with a total of 161 attributes, 31 operations, 71 associations, and 90 generalizations. Since the model is too large to be described and discussed in its entirety here, I am focusing on the areas that were already addressed in the section on requirements analysis: products and contracts.

In order to allow for modeling the ever increasing variety of product types, we decided for overloading classes in the model in a way that they are supposed to represent metaclasses, which are instantiated into objects representing product types. Fig. 1 shows an excerpt of the original object model that focusses on physical products. Financial products are represented in another part of the model. A particular product is represented as an instance of **Product**. It refers either to an object that represents the corresponding product type, or to an instance of **ProductConfiguration** which serves the representation of a certain kind of configuration represented as an XML string. The abstract class **AbstractProduct** serves as superclass of further classes that represent financial and insurance products. The classes **PhysicalProduct** and **ProductConfiguration** are associated with corresponding contract types, which in turn refer to demand forums.

Even though we knew that there may be a considerable variety of contract types, we decided for a pragmatic solution. Our analysis indicated that, at first, a few contract types would be sufficient. These were directly represented in the model. Their commonalities

were captured in a common superclass. These contracts were not the kind that are entered into with individual customers or suppliers. Rather, they were intended as standard agreements that must be accepted by parties wishing to conduct business with or through the platform. The various contract types, such as sales contract, delivery contract or broker contract were represented as subclasses of the common superclass **Contract**.

#### 3.2 Assessment

At first, the assessment of the original data model, which can be presented here only in part, is restricted to the original requirements. The requirements represented a concession to the relevant boundary conditions in two respects. First, we had to take into account the limitations of the available implementation languages; second, we were compelled to keep the implementation effort within reasonable bounds. The assessment is shown in Table 1. Subsequently, the requirements themselves are evaluated with respect to possible modifications which are suited to improve the design and use of the corresponding information system.

With regard to the evaluation of the object model against the original requirements, the following limitations are particularly severe: The specification of new product types suffers not only from the fact that it jeopardizes integrity. In addition, each new product type has to be specified from scratch. There is thus no reuse, which not only limits productivity but also impedes a coherent representation of product types. If the values of particular properties – for example, the color of a car – may vary, this requires elaborate configuration, rather than simply assigning the corresponding state to an instance of **Product**. The creation of product configurations is not supported by the model at all, because neither the variability that characterizes a configuration space nor the associated configuration rules are represented in the model. Instead, configurations are specified as XML files. Supporting the creation of configurations would therefore require an additional tool, for example one based on XML. Such a tool, however, would involve a representation existing independently of the model, which would be expected to give rise to substantial synchronization problems.

The modeling of contracts suffers not only from being restricted to a fixed number of contract types. Since contract types are represented as objects at M0, they cannot be further instantiated. This is a serious limitation in those cases where individual contracts are required. Similarly, the specification of financial products like those offered by insurance companies or banks is represented by a large generalization hierarchy that is limited to properties that apply to product types, thus, leaving no room for representing specific details of product instances like a particular loan.

What could be done to overcome these weaknesses? At first glance, designing a DSML for the specification of product types and of contract types may seem like a promising approach. However, a DSML that is based on a MOF-like architecture would allow for the specification of types only. The knowledge that may exist about specific features of corresponding instances could not be included in the DSML, which would result in the need for redundant specifications. For this reason, MLM seems to be a more suitable approach.

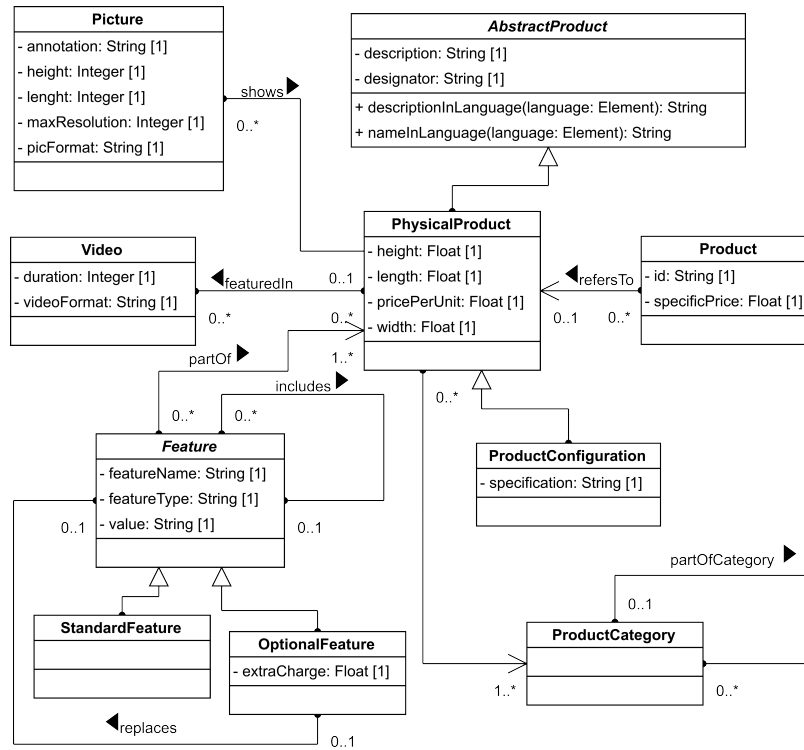


Figure 1: Excerpt of data model: focus on products

#### 4 Reconstruction as Multi-Level Model

With regard to the scope of the model, the reconstruction of the data model at hand is primarily aimed at addressing the requirements outlined above in a more convincing manner. In addition, the case study also includes the reconstruction of the remainder of the model. The complexity of the task recommends the use of a method for the systematic transformation of a flat model into a multi-level model. Note that I use a special terminology for expressing multi-level concepts that was inspired by [10] and is described in more detail in [3]. Instead of “instantiation”, I speak of “concretization” in cases where traditional instantiation is accompanied by inheritance. To avoid confusion, I use “Ln” for marking an object at level n instead of “Mn” known from the MOF. Properties of classes can be assigned an instantiation level to specify deep instantiation.

##### 4.1 A Method for Reconstructing Flat Models

The following method is intended to support the reconstruction of a flat data or object model as a multi-level model. It is initially aimed at identifying those parts of a given flat model that are particularly suitable for transformation into a multi-level model, i.e., that “smell” [7] in that sense. In addition, it supports the construction of the corresponding multi-level model. The method consists of two main components. A process model serves as an idealized point of reference for the stepwise transformation, while analysis and design principles included in the process model support the decisions to be made during the transformation. The method was developed on the basis of a method for designing multi-level models [3] and

builds on previous work by de Lara et al [7]. The application of the method requires careful analysis of the flat models, since it cannot be ruled out that the models contain errors or misleading conceptualizations. The procedure outlined below should not be viewed as a strict sequence. Rather, the criteria emphasized in the individual phases may also be examined iteratively and in alternation.

- (1) *Search for potential type-object patterns*: Associations between classes, instances of which seem to represent classes and classes that are instantiated into objects representing instances are of primary interest, since they indicate the need for introducing classes at L2. In the simplest case, names of corresponding classes and associations are characterized by self-explanatory labels (such as “\*Type”, “instance of” or “type of”). In the original model, this pattern is, for example, represented by the classes **Creditcard\_Type** and **Creditcard\_Spec** the association **cardType**.
- (2) *Search for potential metatype-type patterns*: This pattern is used to allow for the dynamic construction of classes (which are in fact represented by instances) by adding features (attributes, operations ...). Possible indicators are class names including strings such as “Feature” or “Property”. With regard to the case study, it is evident from the outset that the modeling of products is based on this pattern (see Fig. 1) – and includes a type-object pattern, too.
- (3) *Search for potential variations of types/classes*: This activity is particularly relevant with regard to promoting reuse and adaptability. If, in a given context, there is only a single type or class of a certain kind, abstraction to a metaclass is not

**Table 1: Requirement Satisfaction**

| Req.   | satis. | Comments                                                                                                                                                                                                                                                                                                                                                                                  |
|--------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Req. 1 | 2      | Product types are not implemented as types or classes, but as objects at M0.                                                                                                                                                                                                                                                                                                              |
| Req. 2 | 2      | Objects representing particular product instances can be characterized by an id and a production date. However, they are not true instances of classes representing corresponding product types.                                                                                                                                                                                          |
| Req. 3 | 1      | This requirement is addressed only by providing for a representation of a certain configuration through a string, e.g., an XML document.                                                                                                                                                                                                                                                  |
| Req. 4 | 2      | Since product types are represented by objects, they can be added at runtime.                                                                                                                                                                                                                                                                                                             |
| Req. 5 | 2      | The model allows for a high degree of detail, since attributes can be added as desired. However, operations and specific constraints cannot be added at runtime.                                                                                                                                                                                                                          |
| Req. 6 | 0      | Users can add all kinds of meaningless attributes, which would lead to invalid product types.                                                                                                                                                                                                                                                                                             |
| Req. 7 | 1      | The model includes a predefined set of contract types, which can be instantiated into particular contracts. However, this solution suffers from a confusing concept overload. While the creation time clearly corresponds to a particular contract, the validation period might rather represent a feature of the respective contract type.                                               |
| Req. 8 | 1      | The search for products exhibiting specific properties is possible, but highly laborious, since product properties are not represented at the type level, that is, not in the schema. The difficulty is compounded by the fact that, although the attributes specified by <b>Feature</b> can be characterized by data types, the actual data are always represented as character strings. |
|        | 3      | fully satisfied                                                                                                                                                                                                                                                                                                                                                                           |
|        | 2      | fully by workaround                                                                                                                                                                                                                                                                                                                                                                       |
|        | 1      | partially by workaround                                                                                                                                                                                                                                                                                                                                                                   |
|        | 0      | not implemented                                                                                                                                                                                                                                                                                                                                                                           |

indicated. If, however, multiple variants of a type exist or might exist in the domain of interest – as is the case in the original model, for example, with contracts – abstraction to metaclasses may be advantageous, provided that generalization is not sufficient (see below). Possible indicators include class names like “feature” or “category”, or overloaded generalization hierarchies (see below). Attributes used to distinguish types are suspicious, too. For example, an attribute like **pubType** to distinguish different kinds of publication types.

- (4) *Identify potential states of classes*: Classes may define properties that on closer inspection are not characteristics of its instances, but genuine class properties. Examples include the date, when a class, e.g., representing a certain type of contract, was created, or the institution that promoted the type. One possible indicator is the use of attribute names that suggest a date or an average value. However, an attribute

that indicates a date might also apply to the instance level. In addition, it may be the case that all instances of a class exhibit the same values for a particular attribute (or the same references for a particular association). In such cases, it is natural to regard the corresponding feature value as a characteristic property of the class, not least in order to avoid redundant replication across the set of instances. Note that assessing this criterion requires the analysis of larger sets of instances. Data at M0 is also useful for assessing the other criteria (see also phase 6). In case data at M0 are not available, it is an option to create (or generate) a set of example instances.

- (5) *Analyze specialization/generalization hierarchies*: Specialization is used to distinguish between types (classes) of a similar kind. This applies especially to classes located on the same level in the generalization hierarchy. In the original flat model this is the case for modeling financial products (referred to as “service products”). To analyze whether an additional level would be preferable over generalization, it is essential to check, whether it is required to represent instances in addition to types (such as a specific insurance contract in addition to an insurance type). For closer examination, the criterion focused on in phase 4 needs to be accounted for.
- (6) *Search for datatypes representing structured data*: Such data formats may point to deficits in the abstraction capabilities of the modeling language used. In the original model, this applies to the configuration of products, which is reduced to the attribute **specification** of **ProductConfiguration** and represented solely in an XML structure. This kind of smell is easy to detect in case the original modeling language supports respective datatypes such as “XML\_String”. Otherwise, it is required to check corresponding instances, if these exist.
- (7) *Analyze data at M0*: To capture additional classification options, questions must be asked that concern possible abstractions over sets of instances. For example: In which cases would it be useful if statistical evaluations (e.g., mean, variance, etc.) of particular properties of the instances of a class were available? These could then be specified as operations in metaclasses and executed by the respective class. Can a set of instances be meaningfully partitioned into homogeneous clusters, for example by means of factor analysis? If so, such clusters could, under certain circumstances, be specified as classes of a common metaclass. Due to the lack of a significant amount of data at M0, this was not an option in this case study.
- (8) *Reconstruct affected parts of the model*: For this purpose, the guidelines proposed in [3] are particularly relevant. If the introduction of an additional level is a viable option, it must be examined, on the one hand, whether the concretization of a corresponding class on Ln into classes on L(n-1) promises benefits in the context of model use. On the other hand, it must be assessed whether the corresponding abstraction is temporally invariant, since the tight coupling associated with it would otherwise impede the future adaptation of the model. The same examinations are to be applied to the

newly designed classes on  $L_n$  in order to find out whether further classes on  $L_{(n+1)}$  make sense. In this way, a multi-level hierarchy gradually emerges. For all properties of the classes in a concretization hierarchy, it must be determined at which level they are to be specified first and at which level they should be instantiated. The guidelines provide only a general orientation, one that requires consideration of further questions, such as “How can it be determined whether an abstraction is invariant?” or “How can the appropriate instantiation level of a property be identified?”. An examination of these questions can be found in [3].

The proposed criteria should not be considered in isolation. For example, the assessment of a generalization also depends on the nature of the variation within individual classes and whether it makes sense to assign a state to a class. Also, if a possible variety of a class is identified, this may be associated with different class states.

## 4.2 The Multi-Level Model

FMML<sup>X</sup>[2, 4] and XModeler<sup>ML</sup>[6] were used to reconstruct the original model. XModeler<sup>ML</sup> allows for developing multi-level models using a bottom-up approach by adding new meta classes (levels) during runtime. To take advantage of this feature, the original model was at first transformed to a flat object model in XModeler<sup>ML</sup>. While the reconstruction was primarily aimed at transforming the original model into a multi-level model, individual weaknesses were also remedied. Presenting the complete multi-level model would exceed the scope of this paper. To nevertheless provide a viable understanding of the reconstruction, an overview of the most strongly affected parts of the original model is given first. Subsequently, selected parts of the multi-level model are presented. The complete multi-level model as well as the original object model and the accompanying business process models are available at [www.wi-inf.uni-due.de/LE4MM/xmodelerml/#caseStudy](http://www.wi-inf.uni-due.de/LE4MM/xmodelerml/#caseStudy).

All classes in the original model were grouped into subjects, e.g., products, forums. Each category was assigned the patterns that are possible indications for a reconstruction to be useful. The patterns correspond to the criteria presented with the above method: metatype type (“m”), type-object (“o”), generalization (“g”), class state (“s”), and class variety (“v”). Subjects that seemed suspicious are shown in Table 2. In addition to patterns, subjects are characterized by the number of corresponding classes, and, if applicable, by the depth of the included generalization hierarchy. For the subject categories “customer notification” and “packaging”, it seemed sufficient to do without states of the respective classes. All other subject categories were reconstructed as multi-level models.

Due to space restrictions, I will present only selected parts of the reconstructed multi-level model: physical products and configurations of physical products. Representing an ever growing set of physical product types was regarded a pivotal challenge in the original design. In order to satisfy Req. 3, new product “types” could only be introduced as objects at M0. The transformation of the flat product model into a corresponding multi-level model could have been implemented easily and without significant effort. In XModeler<sup>ML</sup>, every class is an (executable) object. Therefore, new

**Table 2: “Smells” in the original object model**

| subject                | pattern   | depth | classes | changed |
|------------------------|-----------|-------|---------|---------|
| financial products     | g,s,v     | 3     | 16      | yes     |
| notifications          | v         | -     | 3       | no      |
| packaging              | g,v       | 1     | 12      | no      |
| physical products      | m,o,g,s,v | 3     | 12      | yes     |
| accounts               | g,v       | 1     | 2       | no      |
| contracts              | g,s,v     | 2     | 4       | yes     |
| credit cards           | o,s,v     | -     | 2       | yes     |
| product configuration  | s,v       | -     | 1       | yes     |
| finan. product config. | s,v       | -     | 1       | yes     |

classes with any properties can be added during runtime. To support a clear distinction between properties of product types (e.g., weight) and properties of product instances (e.g., serial number), it is useful to introduce a class at L2. Such a class enables the instantiation of type-level properties at L1 while simultaneously allowing the deferred instantiation of instance-specific properties at L0. Such a solution would, however, be unsatisfactory, since it would fail to satisfy Req. 6 and would moreover make only insufficient use of the possibilities offered by a multi-level architecture. Therefore, the reconstruction was aimed at the construction of multi-level DSMLs that facilitate the convenient and safe specification of new product types. If necessary, additional DSMLs could be added, or existing ones could be further concretized.

The class **PhysicalProduct** at L4 in Fig. 2 represents a general notion of physical products. The classes at L3 serve the representation of product categories. Their properties represent knowledge known about corresponding products that is available already at this level of abstraction. These classes can also be regarded as concepts of a higher level DSML. Due to space limitations, only a few properties are shown in the figure. Classes at L3 serve the specification of product lines. A product line is an abstraction over a set of product models that share certain commonalities. Classes at L2 can be concretized into classes representing particular product models which in turn can be instantiated into objects representing particular exemplars. The association between **PhysicalProduct** and **ProductManager** illustrates that associations between classes at different levels are possible. It also shows that violations of constraints are indicated in the diagram editor (the class **Sofa** would have to be associated with an object of class **ProductManager**).

To a certain extent, the model also supports the configuration of products in the sense of a purposeful selection from a set of predefined alternatives. For example, different colors and materials can be chosen for sofas. The constraint **colourMaterial** specifies which combinations of colors and materials are not permitted. However, with regard to more extensive configurations, the model is not sufficient. The model fragment shown in Fig.3 allows for greater degrees of freedom in the specification of more complex configurations. At the highest level, it represents the composite pattern. While the use of this pattern at the highest level seems like the obvious choice, it creates a major challenge that could not be overcome with earlier versions of FMML<sup>X</sup>. The associations needed to represent the pattern require the specification of multiplicities.

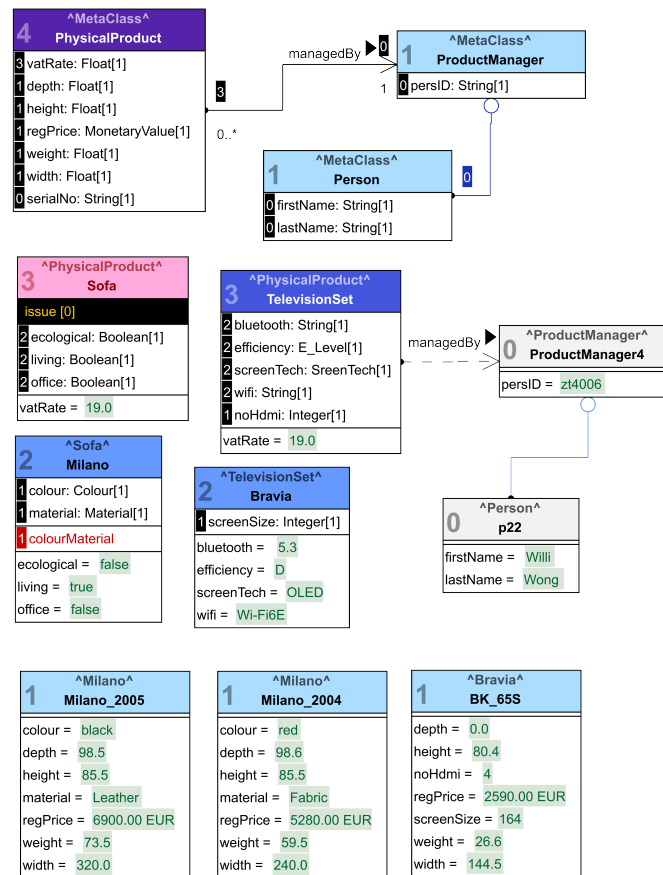


**Figure 2: Multi-level model of products**

However, multiplicities of associations between parts and composites at lower levels may vary. Note that due to space limitations, objects on L0 are not shown.

Newer versions of FMML<sup>X</sup> and XModeler<sup>ML</sup>® allow for the definition of association types [6]. They enable a solution of the multiplicity problem. In the multi-level model of product configurations, association types are defined between **Part** and **CompositePart**, which are instantiated as intrinsic associations at L3. The type of an association is indicated by the color of the edge representing an association. Instantiation of the associations defined at L3 are not restricted to L2, but also deferred to L1 and L0. These instantiations are depicted as dashed lines. In addition to allowing for a relaxed specification of multiplicities, e.g., by using a set of possible values or an interval may be specified, association types offer further features that could be used to create the multi-level model. For example, they allow associations instantiated from them to be named in a way that serves a specific purpose. Furthermore, it is possible to define dependencies between association types, which then apply to the corresponding associations. For example, the association **mayIncludeTire** between **Frontwheel** and **BikeTire** at L3 (to be

instantiated at L1) is dependent on the association **genericTire** (to be instantiated at L2). This implies that a class W representing wheels at L1 may be linked only to those classes representing tires at L1 that are instantiated from classes at L2 that are linked to W. Association types are specified in a dedicated dialog and, in the current version, are not displayed in the diagram; accordingly, no edges appear between the classes at L4.

Both examples illustrate how a system based on a multi-level model could evolve. Higher levels would provide high level, “generic” DSMLs, which allow for a wide range of reuse. These DSMLs would be used to specify more specific DSMLs, with a smaller range of reuse, but offering a higher degree of productivity and integrity. Finally, types (of products and configurations) and even objects at L0 would represent particular types and exemplars.

## 5 Evaluation

A comprehensive evaluation of the multi-level model is not feasible here. I therefore confine the evaluation to selected areas of the model that are representative insofar as they concern constellations that are not uncommon in object models. To this end, the

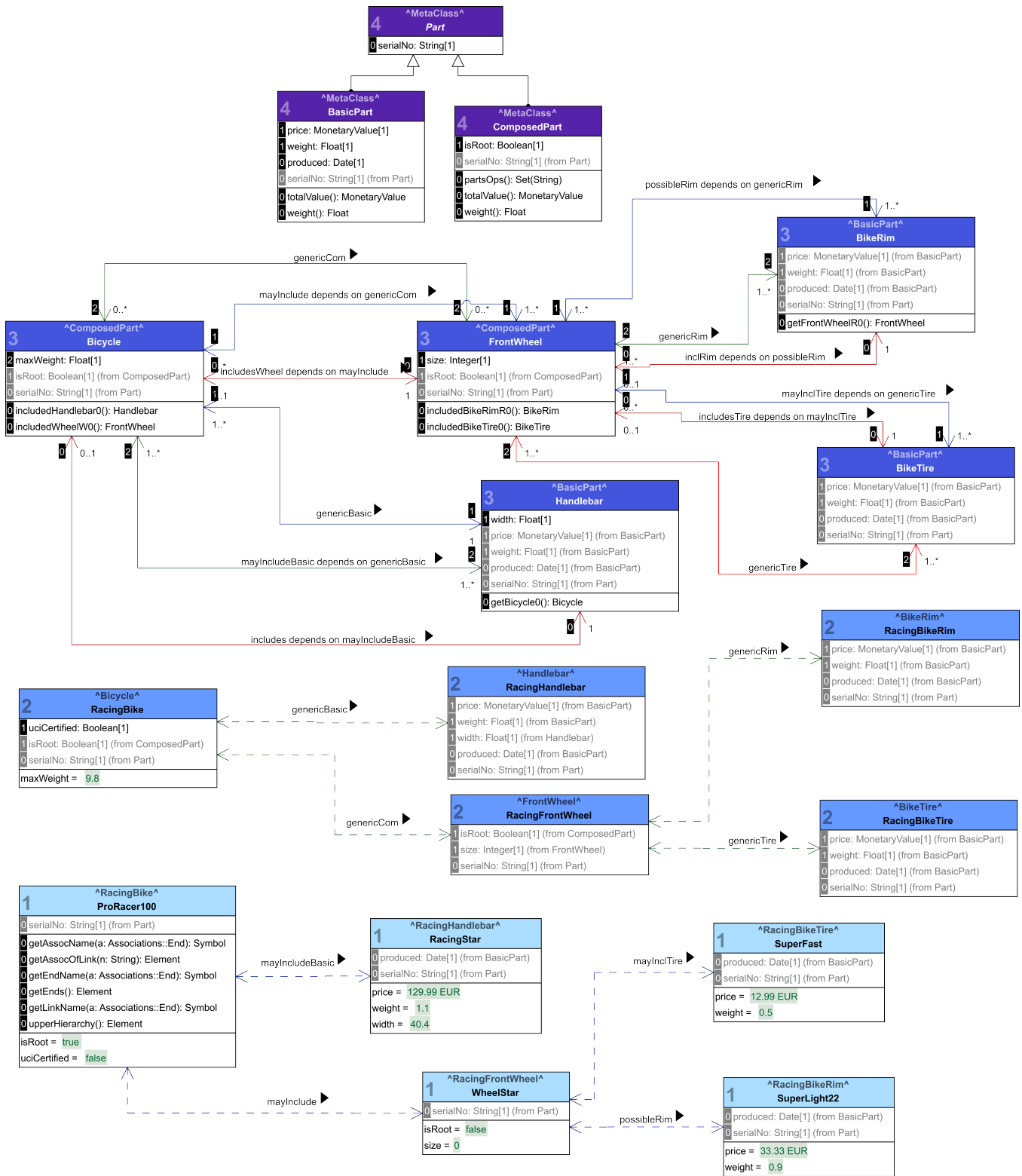


Figure 3: Multi-level model for configuration of physical products

selected areas are first assessed with regard to the requirements. Subsequently, further aspects relating to the use of the model are discussed.

## 5.1 Evaluation against Requirements

At first, the requirements outlined above were extended in a general sense so as to improve reusability, adaptability, and integrity. The following examples illustrate corresponding extensions. It should be possible to create new financial product types at runtime and to specify properties of individual instances, for example a particular credit amount. A credit card should be represented as an instance of one of several possible credit card types, so as, on the one hand, to enable the representation of different credit card types and, on the other hand, to ensure integrity between credit card types and individual credit cards.

*Physical Products:* The multi-level model shown in Fig. 2 is suited to widely satisfy all requirements (for Req. 6 see below). However, it is restricted to two example product categories only. Therefore, the utility of the model finally depends on how it evolves over time. This depends not only on how many product categories or product types are gradually added, but also on the quality of the abstractions involved: only if the abstractions prove to be invariant over time can costly adaptations be avoided while at the same time allowing further instantiations to be carried out conveniently and reliably.

*Configuration of physical Products:* The reconstruction allows to add further configurations at runtime. To further restrict the set of possible configurations, one could add constraints. Similar to the specification of product types, the effort involved in creating a configuration depends on whether more abstract configurations already exist for the corresponding product category.

*Contracts:* The reconstructed model includes an abstract class **Contract** at L2. It can be specialized to classes (also at L2) that serve the specification of certain contract types. For example, a sales contract requires statements concerning warranty terms and right of return (at L1) and should include attributes such as sales price to be instantiated at L0 only. These statements are specified with the classes at L1 that represent contract types. **Contract** also represents knowledge about specific contracts at L1, e.g., attributes to specify the date of signing or associations with contractors, represented by classes at L1.

*Financial Products:* The abstract class **FinancialProduct** at L3 serves the representation of commonalities shared by all products of this kind. It is either concretized or specialized into classes that represent product categories such as **Loan**, **Savings**, **HealthInsurance**, **VehicleInsurance**, etc. Each class at this level serves the specification of properties needed to represent product types at L1 and particular product instances at L0.

*Credit Cards:* The model of credit cards follows a similar pattern. At L3, properties are defined that are required for the specification of credit card categories such as debit card, charge card, revolving card, etc at L2. Specific credit card types offered by particular providers are specified at L1. Particular credit cards, assigned to a person or an organisation, are represented at L0.

*Queries:* The multi-level model supports fine-grained search queries far better than the original model. Different from the overloaded original model, all product features are part of the model and not represented in the data at M0 only.

## 5.2 Comparison

Compared with the original object model, the multi-level model exhibits obvious advantages with respect to the requirements. The selected metrics presented in Tab. 3 represent a further aspect of this comparison. Although the figures – especially those for the lower levels – are of limited informative value, since additional classes will presumably be added over time at those levels, they nevertheless indicate a clear tendency. While the total number of classes increased from 100 to 115, the number of classes on L1 decreased. More important: the number of slots at levels above 0 indicates the reduction of redundancy achieved with the multi-level model. In those parts of the model directly affected by the transformation, this effect is even more pronounced. More important, however, is the fact that the concretization of classes on L1 from classes on L2 is considerably more convenient and reliable than specifying those classes from scratch.

Table 3: Selected Metrics

| Level | Classes | Attributes | Slots |
|-------|---------|------------|-------|
| 1     | 81      | 107        | 67    |
| 2     | 20      | 37         | 28    |
| 3     | 10      | 22         | 2     |
| 4     | 4       | 13         | -     |

## 5.3 Further Aspects

According to Req. 6, users should be prevented from defining product types that are beside the point. This requirement applies also to the construction of other types, like of contracts or services. The effort involved in creating a configuration depends on whether more abstract configurations already exist for the corresponding product category. In this respect, the evaluation of the model is ambivalent.

On the one hand, the convenient and reliable creation of new product types is supported by DSMLs at higher levels, for example by a language that generally structures the modeling of television sets (indicated in the example by the class **TelevisionSet**). The more specific the predefined classes are, the stronger this support becomes and the less likely it is that users will define nonsensical properties. This advantage of multi-level models must, however, be viewed in context. Users at higher levels do not receive such support and are therefore required to define appropriate properties on their own. In addition, it is in principle also possible at all levels above L0 to add arbitrary attributes, operations, and associations.

This is due to the recursive nature of the meta model, FMML<sup>X</sup>, is based on (see, e.g., [6]): every class at any level can be assigned an unlimited number of properties. This creates a conflict between flexibility and integrity. At higher levels, it will usually be no option to restrict the model/language designers freedom. For the design

of these levels, the involvement of qualified modelers or language engineers is therefore recommended. At lower levels, users can be prevented from making certain changes. This can be achieved by hiding certain features or by adding specific constraints [5].

## 6 Related Work

De Lara et al. [7] analyze a large number of metamodels and identify various patterns (“smells”) that recommend a multi-level reconstruction of models created with these metamodels. The “smells”, including the type-object pattern and the metatype-type pattern (referred to as “dynamic type”), correspond in part with those used in the method presented here. In addition, they accounted for patterns like powertype and stereotypes.

Building on previous work by De Lara et al., Macías et al. [8] aim to support the automated transformation of flat models into multi-level models. To this end, they propose enriching metamodels used for the specification of flat modeling languages with annotations that indicate possible “smells” that may suggest the need for reconstruction. Their investigation appears to focus on metamodels of DSMLs. The process model they propose provides for transforming correspondingly annotated flat metamodels into suitable metamodels of multi-level languages. This approach is not directly applicable to the present case study, because no corresponding patterns would be identifiable in the metamodel of the UML-like language. At the level of object models, however, starting with the detection of “smells” corresponds to the approach here.

Neumayr and Schrefl [11] take a different perspective on transforming a flat model into a multi-level model. They mainly focus on the question how a multi-level model could be adequately represented by means of a two level modeling language. Instead of using a MLM language for that purpose, they propose to use a pattern, called “Deep Domain Object” (DDO), to represent a multi-level model with a two level modeling language. Thus, it would not be required mapping a flat model to a multi-level model that is created with a multi-level language. Instead, “deepening” of the flat model would not require a further language, but only a redesign of the original model where aggregation relationships together with generalization/specialization serve to represent the type object pattern at various levels. However, this would ultimately mean giving up the specific benefits of abstraction offered by a multi-level language.

## 7 Conclusions and Future Work

The multi-level model developed in the case study exhibits clear advantages over the original model not only with respect to reuse, adaptability and integrity. It also facilitates more fine-grained and, at the same time, more convenient queries. It must be borne in mind, however, that such claims can never be made with respect to multi-level models per se. Rather, it is essential to account for their quality – for example, whether the abstractions they contain are temporally invariant. Also, the analysis of the case study supports the following assumptions (provided that the example model may be regarded as reasonably representative of enterprise systems): The proportion of classes suitable for reconstruction is limited (approximately 20% in the case examined). Nevertheless, even a partial reconstruction as a multi-level model is highly labor-intensive. A

comprehensive analysis of data – that were not available in the case study – may lead to additional possible abstractions.

In the case study, the objective was not limited to evaluating the multi-level model produced, which in any case represents only an early stage in a longer evolution of the model that comprised especially adding classes at L2 and L1. Rather, the study was also intended to develop and test methodical support for the transformation of flat models into multi-level models. In addition, the effort associated with such a transformation was to be examined. The method presented proved effective. Its application, however, requires a considerable degree of domain knowledge and analytical competence. The effort involved in transforming a flat model of this size was substantial, even though the conditions in the present case were favorable: comprehensive documentation was available, and the transformation was carried out by the person who had created the original model.

Probably, a large number of existing schemas and models could be significantly enhanced through a reconstruction as multi-level model. In many cases, however, the effort involved is likely to constitute a serious obstacle. It seems therefore natural for future research to focus on supporting this process through the greatest possible degree of automation. One possible approach would be to train an LLM –for corresponding experiments see [9] – on the basis of the method presented in addition to applying algorithms aimed at the inductive generation of possible abstractions.

## References

- [1] Ulrich Frank. 2002. Modeling Products for Versatile E-Commerce Platforms – Essential Requirements and Generic Design Alternatives. In *Conceptual Modeling for New Information System Technologies*, Hiroshi Arisawa and Yahiko Kambayashi et al. (Eds.). Springer, Berlin and Heidelberg, New York, 444–456.
- [2] Ulrich Frank. 2018. The Flexible Modelling and Execution Language (FMML<sup>X</sup>) – Version 2.0: Analysis of Requirements and Technical Terminology. ICB Research Report No. 66. University of Duisburg-Essen.
- [3] Ulrich Frank. 2021. Prolegomena of a Multi-Level Modeling Method Illustrated with the FMMLx. In *24th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. IEEE.
- [4] Ulrich Frank. 2022. Multi-Level Modeling: Cornerstones of a Rationale. *Software and Systems Modeling* 21, 1 (2022), 451–480. doi:10.1007/s10270-021-00955-1
- [5] Ulrich Frank and Tony Clark. 2022. Peculiarities of Language Engineering in Multi-Level Environments or: Design by Elimination. In *Kühne (Ed.) 2022 – Proceedings of the 25th International*. 424–433. doi:10.1145/3550356.3561544
- [6] Ulrich Frank and Daniel Töpel. 2024. Association Types: Motivation, Specification and Implementation with the XModeler<sup>ML</sup>. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24)*, September 22–27, 2024, Linz, Austria.
- [7] Juan De Lara, Esther Guerra, and Jesús Sánchez Cuadrado. 2014. When and How to Use Multilevel Modelling. *ACM Trans. Softw. Eng. Methodol.* 24, 2 (2014), 12:1–12:46. doi:10.1145/2685615
- [8] Fernando Macías, Esther Guerra, and Juan De Lara. 2017. Towards Rearchitecting Meta-Models into Multi-Level Models. In *Conceptual Modeling*, Heinrich C. Mayr, Giancarlo Guizzardi, Hui Ma, and Oscar Pastor (Eds.). Lecture Notes in Computer Science, Vol. 10650. Springer International Publishing, Cham, 59–68. doi:10.1007/978-3-319-69904-2\_5
- [9] Pierre Maier and Vicky Kadziolka. [n. d.]. Model Deepening with Large Language Models: Insights from Exploratory Studies with ChatGPT. Lecture Notes in Computer Science, Vol. 16190. Springer, 119–136.
- [10] Bernd Neumayr and Michael Schrefl. 2009. Multi-Level Conceptual Modeling and OWL. In *Advances in Conceptual Modeling - Challenging Perspectives*, Carlos Alberto Heuser and Günther Pernul (Eds.). Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 189–199.
- [11] Bernd Neumayr and Michael Schrefl. 2022. Domain object hierarchies inducing multi-level models. *Softw. Syst. Model.* 21, 2 (2022), 587–621. doi:10.1007/s10270-022-00973-7