Modeling Facets of a Warehouse with the FMML^{X}: A Contribution to the MULTI Warehouse Challenge

Ulrich Frank*, Pierre Maier*, Daniel Töpel*,
* *University of Duisburg-Essen*, Essen, Germany

***Abstract*—This paper presents a contribution to this year's MULTI challenge concerned with the modeling of physical products in a warehouse. Based on an analysis of the requirements presented with the challenge, we develop a corresponding multi-level model with the FMML^{X}. The model is implemented with the XModeler^{ML}. Therefore, it is executable. We evaluate the solution against the requirements and discuss it with respect to principle design decisions, its adaptability, and the inspiration we received from this work for future enhancements of the FMML^{X}.**

***Index Terms*—multi-level modeling, warehouse challenge, FMML^{X}, XModeler^{ML}, modeling guidelines**

I. Introduction

Research on multi-level modeling (MLM) has produced a variety of multi-level modeling languages and respective modeling tools (e.g., [1], [4], [8], [10], [17], [20], [23], [25]). Apart from common foundational concepts, these approaches differ both in terms of particular language terms and with respect to corresponding modeling tools. In recent years, there has been a clear interest in consolidating research. This is expressed above all in promoting the comparability of the different approaches and, thus, defining a common core.

The MULTI challenges series reflects this objective. A challenge relates to a use case, which is described in terms of requirements that need to be satisfied by the respective MLM approach. This year's challenge concerns the management of physical products in a warehouse [21].

Products represent an especially suited subject for MLM because they exist in a considerable variety, which demands for powerful abstraction concepts. It is therefore not surprising that the representation of products often serves as a prime example to motivate the need for MLM (e.g., [2], [3], [8], [10], [19]).

Before we present and discuss our solution, we introduce parts of a multi-level modeling method we applied to develop the model. It consists of the FMML^{X} (section II) and guidelines that support design decisions (section III). In addition, we provide a short overview of the corresponding modeling tool, the XModeler^{ML} (section II), and an analysis of the case described with the challenge (section IV). The main part of the paper is dedicated to the design of the solution (section V). Subsequently, the solution is evaluated against the requirements (section VI). At last, we discuss additional aspects of the presented model's quality and of the FMML^{X} (section VII).

II. Background: FMML^{X} and XModeler^{ML}

The flexible multi-level modeling and execution language (FMML^{X})[1] is an object-oriented language and provides the following basic modeling constructs. *Classes* are intensionally defined by their properties, that is, attributes, operations, associations and constraints. Attributes are defined by a class or an enumeration type and a name. Each class is an object at the same time. Hence, it has state, may execute operations, and is an instance of a class. As a consequence, the FMML^{X} allows for an arbitrary number of levels. We do not use the common term "clabject" (e.g., [3]) to express this duality, since the metamodel of the FMML^{X} makes an explicit distinction of the concepts *Class* and *Object* (see below). To avoid confusion, we speak of "class" whenever we refer to the class facet of an object, otherwise of "object."

Each object in an FMML^{X} model is assigned an explicit level. A level indicates the classification level, but must not be confused with pure classification (at least in case of the classification of classes). A class is created from its class through an act we call *concretization*, following a proposal in [24]. Concretization is different from instantiation because a concretized class does not only instantiate properties defined with its class but also inherits from the root class of the core metamodel (see below). Each instance or concretization of a class A is called its *descendant*; class A inversely the *ancestor*. We call the set of all descendants of a class the *concretization tree* of that class.

As a default, the level of a class reflects the number of classifications starting at L0 (we use "L" as an abbreviation for "level" in the following). Properties can be defined as intrinsic, which means they are supposed to be instantiated not within direct descendants of a class, but only further down the concretization tree ("deferred instantiation"). To that end, the specification of intrinsic properties includes the definition of the level where they should be instantiated. This *instantiation level* of an intrinsic property must be lower than the level of its class minus one. To facilitate consistent changes, intrinsic properties are specified in a FMML^{X} model only once. Within lower level classes only include references to these specifications.

Associations may be defined between two classes at different levels. However, generalization/specialization relationships can be defined only between classes at the same level.

[1]Note that the acronym was resolved to "flexible meta-modeling and execution language" in early publications.

979-8-3503-2498-3/23/$31.00 ©2023 IEEE
DOI 10.1109/MODELS-C59198.2023.00107

Operations and constraints are defined with the executable object constraint language (XOCL)[2]([5], [6]). In other words: FMML^X models are represented with the XOCL, a complete programming language, which makes them executable. The FMML^X comprises a default notation that follows the UML style and, among others, uses colors to distinguish levels. To customize the design of multi-level DSMLs, the default notation can be replaced by a domain-specific variant.

The FMML^X is specified as a monotonic extension of "XCore," the meta-model of the executable meta-modeling facility (XMF) [6]. Since it implements a "golden braid metamodel architecture" [5, pp. 23f.], XCore enables an arbitrary number of classification levels. The core idea is that each class (except for classes at L1) inherits from `Class`. Hence, these essential properties are available with every class, which corresponds to the *linguistic metamodel* other MLM approaches are based on.

At the same time `Class` inherits from the class `Object` (which itself is an instance of `Class`) with the effect that every class has object properties, that is, it has a state and can execute operations. Different from the FMML^X, XCore does not provide for the definition of levels or for deferred instantiation.

The XModeler^{ML} extends the XModeler, the implementation of XMF [5]. It implements the FMML^X and allows the execution of FMML^X models. Upon the specification of attributes and associations, the XModeler^{ML} generates corresponding access methods, which can be faded in and out on demand. This is the case, too, for properties that are derived from higher level classes.

The XModeler^{ML} allows to create and interact with FMML^X models by using a diagram editor, a multi-level object browser, an object workspace, and a generated default GUI for selected objects. A concrete syntax editor enables the creation and integration of graphical notations.

The diagrams shown in Fig. 1 and Fig. 3 illustrate core concepts of the FMML^X as well as its default notation. More detailed descriptions of the FMML^X, XCore, and XMF are available in [10] and [5]–[7], [11], [12].

III. Multi-Level Modeling Guidelines

Modeling methods suggested for the use of traditional languages with one classification level only, or for the design of traditional DSMLs (e.g., [9], [18]) provide useful general modeling principles such as "separate invariant from variable parts of a system," or "avoid redundancy." but are not sufficient to guide the appropriate use of additional concepts provided by multi-level languages. We follow the multi-level modeling guidelines proposed in [13] as an orientation for a methodical development of the presented solution. Table I gives an overview of selected design principles that are part of these guidelines.

[2]The XOCL is sometimes also referred to as "executable object command language".

The design principles do not work as a cookbook. Instead, each principle recommends a thorough analysis of the respective domain.

TABLE I
Multi-Level Modeling Guidelines from [13] (Excerpt)

id	Design Principle Description
G-1	Specify known knowledge on the highest possible level within the scope of your project.
G-2	The higher the level of an object, the more invariant it should be.
G-3	The design of an object at any level should aim at modification consistency.
G-4	Assign properties of objects on levels higher than L1 to categories that indicate semantic differences.
G-5	Every class should be assigned the level where it conceptually belongs.
G-6	Avoid the introduction of "fake" levels, that is, of levels that could be expressed through generalization/specialization.
G-7	Commonalities should be captured by an appropriate abstraction also in cases of incomplete knowledge

IV. Warehouse Case Analysis

We analyzed the challenge description to extract requirements and assigned a unique identifier to each one to facilitate their referencing. The analysis showed that some of the requirements are not sufficiently precise or complete. Cases where our interpretation of a requirement may deviate from the authors' intention are marked with an *i*. Additional requirements that follow from the challenge but are not made explicit there are marked with an *a* in table II. To structure the analysis, we start with focusing on different product categories. Subsequently, we will take a closer look at product details.

(1) *Categorization of Products*. The challenge distinguishes between two principal kinds of products. On the one hand, there are product exemplars (copies) that have an identity of their own within the realm of the warehouse. We call this kind of products *identity product* (R-1). An identifier may be explicitly assigned, e.g., through a serial no., as it is exemplified for a DVD player. On the other hand, the warehouse does not keep track of individual exemplars in case of *Bulk products* (R-2).

Bulk products and identity products represent special cases of a more general notion of product ("adhere to the same stipulation") (R-3). Bulk products may be sold in packs. We conclude this from the example of 10-pack batteries (R-4). We also conclude from this example that packs of bulk products (a) qualify as a specific kind of product and (b) do not have an identity of their own in the realm of the warehouse (R-5). Product copies and product specification types conform to their respective types (R-6, R-7).

(2) *Product Pricing*. All products must be assigned a standard sales price (SSP) (R-8). Identity products may have assigned a reduced price that must be lower than the standard price (R-9). All products of any kind may be assigned a minimum price. This property should apply to identity and bulk products alike. However, we assume that it relates to

the minimum standard price for bulk products (R-10) and the minimum reduced price for identity products (R-11).

The currency used to specify any of the above prices is defined at the level of a product specification type. Note that this also covers the statement "Copies conforming to the same product specification are always sold in the same currency" (R-12). Price assignments should not use a currency different from the one defined for their respective product specification type ("type safe") (R-13).

According to the challenge, each product specification type defines a tax rate (R-14). The tax rate is the same for all product specification types (15%), except for books (7%) (R-14). We assume that these reference tax rates are subject to change. The gross price ("final price") of each product is the standard or reduced price plus tax (R-16).

(3) *Product Recommendations*. Products may point to other products to express recommendations. We assume that recommendations of this kind are restricted to product specifications (R-24). A product specification may only recommend other product specifications if they have explicit clearance for this, e.g., if they are compatible (R-25). Since a product should not directly recommend itself, we add requirement (R-26).

(4) *Warehouse Management*. According to the challenge, the warehouse "needs to keep track of the sum of all products sold". We regard this requirement as problematic for two reasons. First, to keep track of sold amounts of bulk products, one would need to introduce further concepts such as invoice, invoice item, etc. We assume that these would be beyond the scope of the warehouse. Second, while it would be possible to mark objects representing specific exemplars of identity products as sold, that would would create serious drawbacks. To support a purposeful analysis of sold products, one would have to add the date when a product was sold. Over time, prices are likely to change. Therefore, one would also have to store the corresponding price history. For these reasons we took the liberty to interpret the requirement as follows: "The warehouse needs to keep track of the value represented by the products it stores, both at the product specification level (R-17) and the product specification type level" (R-18). In case of an identity product, the value of a particular exemplar is determined by its final price. In case of a bulk product, the value is given by the standard price times the amount in stock.

This requirement implies that for all bulk products the amount in stock needs to be recorded. Since we assume that not all bulk products are sold necessarily in packs, this demands for recording both, the amount of unpacked bulk products and the amount of bulk product packs (R-19).

The warehouse management system should be able to "dynamically" add and remove product specification types (R-21). The challenge states that "Product copies may have properties such as 'open box', 'accessories missing', 'returned on 23 March 2023". We interpret this requirement as "It must be possible to add properties for a more elaborate description of product copies." (R-22). This is a special, less demanding, case of the previous requirement. It should be possible to keep track of the date a new product specification type was added (R-23).

(5) *Focus on Particular Product Specifications*. Among others, the warehouse includes DVDs and books. According to the challenge, a book copy is an instance of a book specification that contains bibliographic data such as title (and presumably the name(s) of the author(s) in order to distinguish between two books that share the same title). Furthermore a book specification should define a reference currency and standard sales price. Similarly, a DVD containing a movie is regarded as an instance of a DVD type that includes the title of the represented movie. Note that taking these requirements literally may lead to redundancy, e.g., a paperback and a hardcover representing the same monograph would both include a string to express the same title.

Note that we did not explicitly mention all examples in the challenge, since they are not essential for addressing the requirements. We hope that the solution is sufficiently clear about how they are represented.

V. Model Presentation

The following presentation is divided into four subsections. First, we introduce generic domain classes that capture domain knowledge at the highest level of abstraction (subsection V-A). In the light of guideline G-1, we aim at addressing as many requirements as possible in this subsection already. Thereafter, we focus on the specific categories *identity products* (subsection V-B) and *bulk products* (subsection V-C). Finally, we present a general warehouse management class that allows for multiple kinds of warehouse analysis (subsection V-D). Note that the description does not explicitly account for every requirement. In cases where the satisfaction of a requirement is obvious, we confine ourselves with its description in Table III. With respect to space limitations, we cannot present the specification of all constraints and operations.

In addition to this presentation, the complete and executable model as well as a screencast that demonstrates its use are available at https://le4mm.org/multi-23/. The XModeler^{ML} can be downloaded from the webpages of our project LE4MM [14].

A. Generic Domain Knowledge

The generic domain knowledge is represented by five classes on L3 (see Fig. 1). `Product` specifies the properties common to all kinds of products (R-3). The classes `IdentityProduct` and `NonIdentifiableProduct` are specialized from `Product` and represent generic knowledge that distinguishes non identifiable products (bulk products) from identity products. The class `NonIdentifiableProduct` serves as abstract superclass of the classes `BulkProduct` and `BulkPackage`, which both have in common that they are not identifiable (this is what we conclude from the challenge and the common handling of products like battery packs). We will take a closer look at this conceptualization below.

While it might, at first sight, appear to regard `IdentityProduct` and `NonIdentifiableProduct`

TABLE II
LIST OF REQUIREMENTS

id	Description	
R-1	There are products, exemplars (copies) of which have an identity of their own within the realm of the warehouse.	
R-2	There are also *bulk products*, for which the warehouse does not keep track of individual exemplars.	
R-3	Bulk products and identity products represent special cases of a more general notion of product ("adhere to the same stipulation").	
R-4	Bulk products may be sold in packs.	*i*
R-5	Packs are bulk products, too.	*i*
R-6	Product copies conform to a product specification).	
R-7	Product specifications conform to a product specification type.	
R-8	All products must be assigned a SSP.	
R-9	Identity products may have assigned a reduced price which must be lower than the standard price.	
R-10	Bulk products may be assigned a minimum price that restricts their SSP.	*i*
R-11	In the case of identity products, the minimum prices limits the reduced price.	*i*
R-12	A reference currency is defined with each product specification type.	
R-13	Currencies used for price assignment should be type safe.	
R-14	Each product specification type is assigned a tax rate.	
R-15	There are only two different tax rates: either a standard tax rate (15%) or a reduced tax rate (7%, e.g., for books).	*i*
R-16	The final price of each product is its regular sales price plus tax.	
R-17	It should be possible to calculate the total value of all products that conform to a product specification.	*i*
R-18	This should be possible, too, for all products that conform to a product specification type.	*i*
R-19	The amount in stock needs to be recorded for bulk products.	*i*
R-20	The warehouse needs to be able to iterate over all copies and bulk products it currently has in stock for inventory purposes.	
R-21	The model should allow for adding new product specification types.	
R-22	It must be possible to add properties for a more elaborate description of product copies.	*i*
R-23	The date a new product specification type was added needs to be stored.	
R-24	Product specifications can recommend other product specifications.	
R-25	Recommendation between product specifications are only allowed if this was explicitly defined as possible.	
R-26	Product specifications must not recommend themselves directly.	*a*

as concretizations of `Product`, a closer look shows that there is no property defined by `Product` that is instantiated within one of the two other classes. Therefore, regarding them as concretizations of `Product` would violate guidelines G-5 and G-6.

An L2 descendant of `Product`, or one of its subclasses, corresponds to a product specification type, an L1 descendant to a product specification, and an L0 descendant to a particular product copy. As a result, each product copy is an instance of a product specification (R-6) and each product specification is a concretization of a product specification type (R-7) This allows us to add and remove product specification types as concretizations of `BulkProduct` or `BulkPackage` (R-21). Note that these modifications are, however, restricted to the available classes on L3. We discuss this aspect in more detail in the discussion section (see section VII).

The properties defined with `Product` allow for expressing various kinds of prices that apply to product specifications (R-8, R-10), as well as to the definition of a reference currency R-12 (see below) and the date a product specification was introduced (R-23).

For modeling currencies, the FMML^X provides the auxiliary classes `MonetaryValue` and `Currency`. An object of the class `MonetaryValue` contains an object of `Currency` that represents a currency together with an amount that is specified as `Float`. A particular currency is presented within a diagram as a string, which is an element of an extensible set of currency strings that follow ISO 4217. The class `MonetaryValue` also provides for converting amounts from one currency to another, which, e.g., allows for adding two amounts together that are represented in different currencies.

The attribute `currency` of type `Currency` addresses requirement R-12. This allows each L2 descendant of `Product`, hence, each product specification type, to specify a different currency. A currency string serves as a reference to a web service that provides current exchange rates. The attribute `standardPrice` in `Product` is instantiated on L1 and thus enables each product specification to define a different SSP (R-8). The intrinsic constraint `CurrencyMatch1` in `Product` ensures that the currency of each `standardPrice` on L1 corresponds to the currency specified in the respective product specification type on L2 (R-13):

```
Context Product L1
@Constraint currencyMatch1
    self.standardPrice.currency.abbreviation = self.of().
     currency.abbreviation
fail
    self.name + "'s price must be in " + self.of().currency
     .abbreviation
end
```

Note that the class defined as the context of an intrinsic constraint serves as an abstraction of the specific classes or objects at the level the constraint applies to.

A reduced price can only be defined for product copies (R-9). This requirement is addressed by the intrinsic attribute `reducedPrice` with an instantiation level of 0 within the class `IdentityProduct`. The intrinsic constraint `reducedPriceSmaller` defined in the same class ensures that the reduced price defined on L0 is lower than the SSP assigned on L1 (R-11):

```
Context IdentityProduct L0
@Constraint reducedPriceSmaller
  if self.reducedPrice = null
    then true
  else
    self.reducedPrice.getAmount() < self.of()
    standardPrice.getAmountIn(self.reducedPrice.currency)
  end
fail
    "reducedPrice must be less than standardPrice"
end
```

The amount represented by a MonetaryValue value is retrieved via the getAmount() function. It returns the converted amount if the currencies do not match. Since the definition of a reduced price should be optional (R-9), the multiplicity of reducedPrice has a lower bound of 0 and the constraint reducedPriceSmaller must first check whether the value of reducedPrice is null. Additionally, we must ensure that if a value is provided for reducedPrice on L0, it has the same currency as its product specification type on L2, referred to by applying the method of() twice (it returns the class of the corresponding object) (R-13):

```
Context Product L1
@Constraint currencyMatch0
  self.reducedPrice = null orelse self.reducedPrice.
     currency.abbreviation = self.of().of().currency.
     abbreviation
fail
  self.name + "'s price must be in " + self.of().of().
     currency.abbreviation
end
```

The intrinsic attribute inStock defined with NonIdentifiableProduct serves the representation of amounts in stock (R-19) stored with bulk product types at L1. In addition, it includes the intrinsic operations priceAfterTax() and valueInStock() that address requirements R-17 and R-16. IdentityProduct defines the intrinsic attribute id (R-1).

While the classes IdentityProduct and BulkProduct seem like obvious choices, the conceptualization of packs is more demanding. We decided to represent them by the class BulkPackage, which we defined, like BulkProduct, as subclass of the abstract class NonIdentifiableProduct. Packs are products and we assumed that the warehouse does not keep track of single packs, which makes them non identifiable in this context. A pack is characterized by the number of pieces it contains (attribute piecesPerPack). This conceptualization follows the conjecture that bulk packages and the bulk items they contain are different products, and, as such, might define different values for attributes like introduced, taxType, or standardPrice.

This conceptualization represents a simplified version of the composite pattern, since we assume that packs can contain bulk products only, not other packs. Note that, at this level we cannot yet tell much about possible kinds of containment and their properties such as cardinalities.

For tax rates, we distinguish between a 7% reduced tax rate and a 15% standard tax rate. Following G-7, we assume that further tax categories might be added in the future, so a Boolean type cannot be used for this purpose. Instead, we define an enumeration Tax, with the values NORMAL and REDUCED, and add the attribute taxType: Tax in Product to assign a tax type to a product specification type on L2. These values are used in the operation taxRate(), specified in Product. It returns the tax rate according to the specified category and thus fulfills R-15 and R-14.

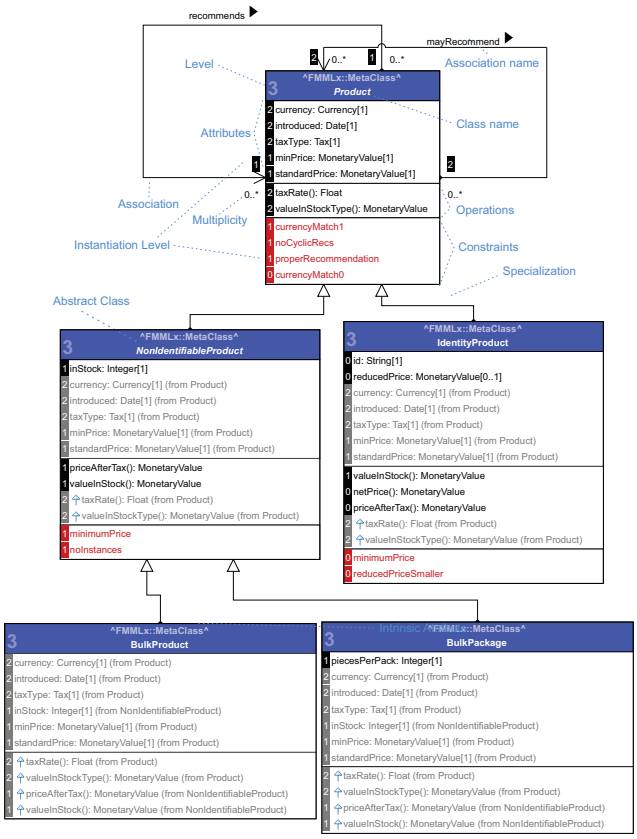

Fig. 1. Product class with specializations on L3

Since the operation is defined only once in Product, the tax rates can be updated and further tax categories can be added with minimal effort. This avoids any redundant specification of tax rates.

While the gross price of bulk products is computed directly by the operation priceAfterTax() specified within NonIdentifiableProduct, calculating the final price of an identity product requires to account for the reduced price, if applicable:

```
Context IdentityProduct L0
@Operation netPrice():Auxiliary::MonetaryValue
  if self.reducedPrice = null
  then self.of().standardPrice
  else self.reducedPrice
  end
end
```

Based on this, the final price is calculated by the operation priceAfterTax().

The accumulated value of products that belong to one product specification (R-18) must also be calculated differently for bulk and identity products. For L1 descendants of IdentityProduct, the accumulated value is calculated by the intrinsic operation valueInStock() based on the priceAfterTax() value returned by the L0 product copies of a product specification (see below).

```
Context IdentityProduct L1
```

```
@Operation valueInStock():Auxiliary::MonetaryValue
  let
    sum = Auxiliary::MonetaryValue(0.0,self.of().currency)
  in self.allInstances()→collect(i |
      sum := sum.add(i.priceAfterTax()));
    sum
  end
end
```

For L1 descendants of `NonIdentifiableProduct`, the sum is calculated based on the `standardPrice` value at L1 by the intrinsic operation `valueInStock()`.

Computing the accumulated value of product specification types (cf. R-18), however, does not require to distinguish between identity and bulk products, since it makes polymorphic use of the two implemenations of the operation `valueInStock()`. Therefore, the corresponding operation can be defined in `Product`:

```
Context Product L2
@Operation valueInStockType():Auxiliary::MonetaryValue
  let sum = Auxiliary::MonetaryValue(0.0,self.currency)
  in self.allInstances()
       →collect(i | sum := sum.add(i.valueInStock()));
    sum
  end
end
```

Further operations to calculate stock values are presented in section V-D.

The intrinsic association `recommends`, which is to link descendants of `Product` at L1, serves to fulfill requirement R-24. The association alone, however, is not satisfactory, since only products of a certain kind may be involved in recommendations (cf. R-25). To address this requirement, we define the association `mayRecommends` that enables to link a product specification type with those product specification type the descendants of which its descendants may recommmend. The constraint `properRecommendation` checks whether a `recommends` link on L1 is proper in the sense that the corresponding ancestor product specification types on L2 are connected via a `mayRecommends` link:

```
Context Product L1
@Constraint properRecommendation
    self.getRecProducts()
        →forAll(p1 | self.of().getRecommendableProducts()
            →exists(p2 | p1.isKindOf(p2)))
fail
    "Some recommended products must not be recommended"
end
```

The constraint `noCyclicRecs` in `Product` serves preventing products from directly recommending themselves. (R-26):

```
Context Product L1
@Constraint noCyclicRecs
    not self.getRecProducts()→includes(self)
fail
    "A product must not recommend itself."
end
```

Bulk products face additional modeling requirements that can be addressed on L3, too. Requirement R-2 prohibits the existence of bulk product exemplars in the warehouse. Accordingly, the constraint `noInstances` within `NonIdentifiableProduct` is to make sure that no bulk products must exist at L0.

```
Context WarehouseManager
@Operation bulkValueInStock():Auxiliary::MonetaryValue
  let sum = Auxiliary::MonetaryValue(0.0,Auxiliary::eur)
  in @For p in Challenge23::BulkProduct.allMetaInstances()
     →select(o | o.level = 1) do
      sum := sum.add(p.priceAfterTax().mul(p.getInStock()))
    end;
    sum
  end
end
```

The `identityValueInStock()` operation follows the same principle. But here the descendants of `IdentityProduct` on L0 are gathered and the value returned by the operation `priceAfterTax()` is accumulated. Both these operations are used in the `totalValueInStock()` operation.

B. Identity Product Descendants

Most requirements concerning identity products are already met by our specification of classes on L3 (see subsection V-A). The class `IdentityProduct` can be used to concretize product specification types on L2 which, in turn, can be used to concretize product specifications on L1. These can then be used to instantiate product copies on L0.

As we indicated already with the analysis of the requirements, the conceptualization of books and DVDs suggested by the challenge may lead to redundancy. Also, from an ontological perspective, there is an obvious difference between a book as a physical artefact and the bibliographic artefact it represents. After consulting with the workshop chairs we were advised to follow the conceptualization suggested by the challenge. While this is certainly acceptable in order to simplify matters, we are not comfortable with designing a model that would violate principles we regard as relevant. Therefore, we decided to develop two variants of the model, which are both available for download at https://le4mm.org/multi-23/.

The default variant is a reflection of what is demanded by the challenge authors. It provides for capturing both, properties of the medium (book, DVD) and the represented content in one product type specification (`Book` and `DVD` respectively). Hence, the object representing a book type at L1 includes pricemarks and the book title, where a reduced price maybe defined with its instances. DVDs are represented accordingly.

The extended variant reflects the idea of distinguishing between an intellectual artefact, such as a monograph or a movie, and its representation in a book or on a DVD. It avoids the redundancy issue described in section IV and corresponds to the default variant as follows. According to the challenge, Moby Dick is a Book Spec that classifies books that comprise copies of Moby Dick. This is represented by the default variant and would translate to “The book type Moby Dick Book classifies books which represent the monograph Moby Dick.” in the extended variant.

In the diagrams showing the model (Fig. 3 and Fig. 2), the parts that were added to the default variant are covered by semi-transparent areas. Note that the attributes `title` in the classes `Book` and `DVD` as well as the corresponding slot values in their concretizations are not required for the extended

variant. Accordingly, the identity product specification type DVD and its descendants on L1 and L0 are described in the default variant with the attribute title, while the extended variant adds the class Movie.

The remaining identity products are shown in Fig. 2. The object MobilePhone, which represents a product specification type, provides reference values to attributes defined in Product that have an instantiation level of 2. For example, MobilePhone defines a currency with the slot currency (cf. R-12) and an introduced date (cf. R-23). Some slots are specific to descendants of IdentityProduct, like the id slot in the L0 object mate08151 (cf. R-1).

In the following, we only describe a few exemplary objects at L1 and L0. The models provided at https://le4mm.org/multi-23/ include all objects and can be easily populated with further instances.

Operations that are executed by these objects return the respective value as defined in the classes Product or IdentityProduct. The object mate08151 returns a net price of 599.15 SEK which corresponds to the SSP in its product specification Mate0815 because mate08151 does not specify a reduced price.

C. Bulk Products and Bulk Packages

An example bulk product specification type from the case description is shown in Fig. 3. BatteryCell is a concretization of BulkProduct. BatteryPack is a concretization of BulkPackage. The association containsBatteries, defined between BatteryCell and BatteryPack, allows battery cells to be contained in multiple package types – or none at all. We also added some battery-specific attributes to BatteryCell like voltage and recharchable. BattSize is an enumeration that contains the values AA and AAA.

Since both L1 objects, EnergeticPlus and EnergeticPlus_Pack10 are descendants of Product, they must both provide values for the different pricing attributes. They therefore define an SSP, which is 1.50 NZD for EnergeticPlus and 12.00 NZD for EnergeticPlus_Pack10. The SSP of EnergeticPlus corresponds to the price value of a single battery cell.

D. Warehouse Management Class

While objects on L2, which represent product specification types, can iterate all objects within their respective concretization tree, no class could iterate all product objects. Therefore, the iteration of all product objects (cf. R-20) is realized through a separate WarehouseManager class (see Fig. 4). We calculate the total value of products in the warehouse by first determining the value of bulk products and identity products separately and then adding both together.

The operation bulkValueInStock() initializes a variable sum of type MonetaryValue at 0.0 EUR. Then, all L1 descendants of NonIdentifiableProduct are iterated and, for each, the value returned by priceAfterTax() is multiplied by the number of pieces in stock. This value is accumulated in the sum variable, where add converts any MonetaryValue to the initialized currency EUR.

VI. Model Evaluation Against Requirements

We consider each requirement shown in table II as satisfied by our solution. Note that some requirements deviate from the original case description as discussed in section IV. Table III summarizes how the requirements were addressed.

VII. Discussion

Although we believe that our solution satisfies the requirements of the challenge, a closer look at the solution reveals certain limitations. The following discussion of principle strengths and possible limitations accounts for the aspects suggested by the challenge except for two: We described and elucidated the basic modeling constructs in section V and outlined how class levels are created technically in section II.

A. Reuse, Adaptability, and Integrity

The rationale for deciding what level a class should be assigned to ("rationale for assigning elements to levels") is reflected in guidelines G-1, G-2, G-3, and G-5. Applying these guidelines is not trivial, though. It requires substantial knowledge of the domain to decide which of its characteristic properties are likely to be invariant over time, and – of course – there is no way of proving that. If the guideline is applied properly (which is, again, not trivial to tell), it will enable an appropriate "balance between prescriptiveness and variability," as it is mentioned in the challenge. If classes at higher levels and their relationships to classes at lower levels are invariant also in the light of new requirements, a corresponding model will support convenient adaptations that maintain the model's integrity.

In that case, adding and removing properties at higher levels will lead to consistent updates of affected elements at lower levels. Due to the fact that the XModeler^{ML} does not support static typing, not every modification can be performed without user interaction. While the design of the solution we present in this paper is solely aimed at fulfilling the requirements of the challenge, a comprehensive evaluation of the model recommends accounting for possible future changes. At first, this relates to changes that may occur within the specific focus of the challenge. It may, for example, happen that bulk products turn into identity products if the costs caused by distinguishing particular exemplars decrease the additional benefit keeping track of every exemplar. Within the current solution, this would require a major change since it would likely involve class migration or a reconstruction of the entire multi-level hierarchy.

With respect to selling representations of artifacts such as movies or monographs, it is likely that other representations such as streaming as well as corresponding pricing models will have to be accounted for at some point. Also, certain products may be bundled with others. For example, batteries may be part of electric devices. In that case, there will be no price

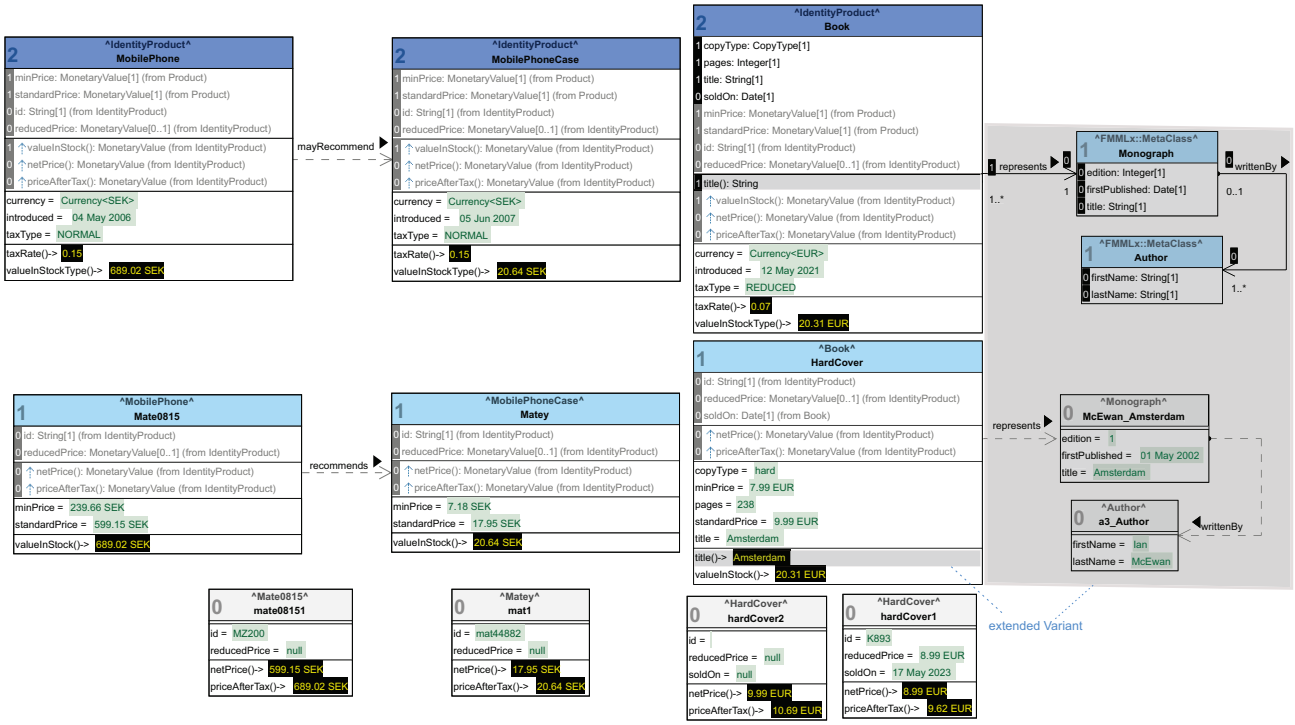

Fig. 2. Descendants of `IdentityProduct`

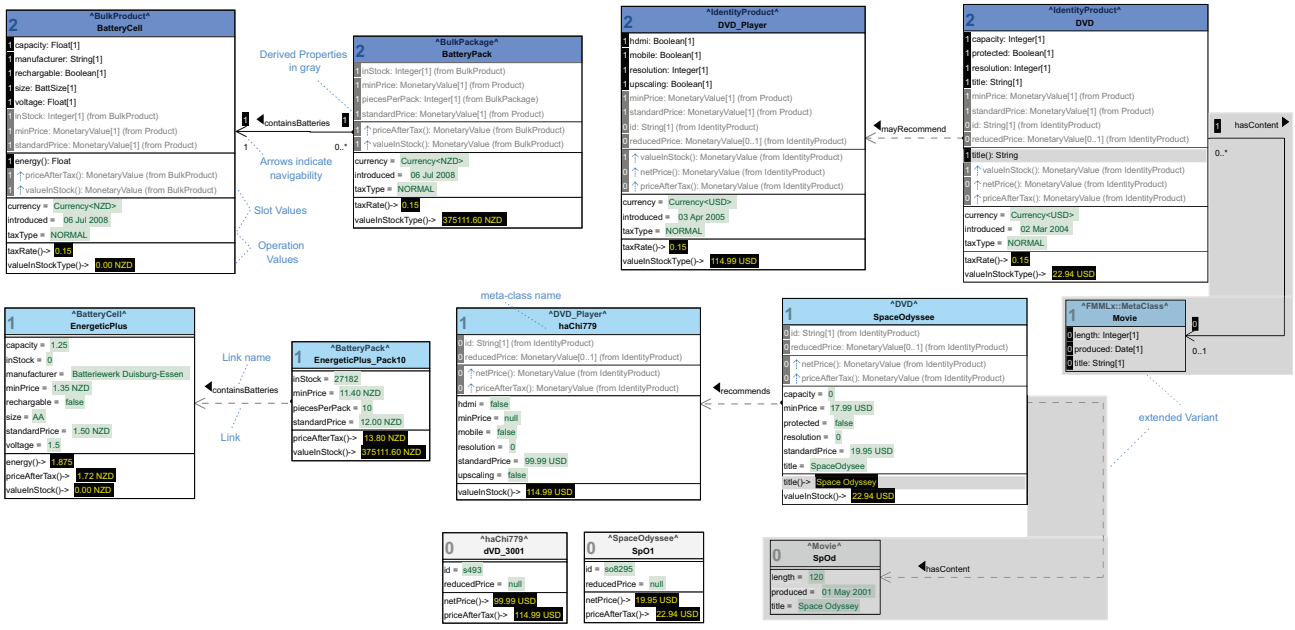

Fig. 3. Remaining descendants of `NonIdentifiableProduct` and `IdentityProduct`

assigned to them. Second, related to requirement R-23, adding new product types may require additional classes at the level of `NonIdentifiableProduct` and `IdentityProduct`, e.g., for perishable foods that need to be refrigerated or for financial products. Furthermore, other product types may require configuration, e.g., picking colours, materials, or extra features. This may lead to the need for additional levels, which, in turn, may require contingent level classes [16].

TABLE III
SUMMARY OF IMPLEMENTATION OF REQUIREMENTS

ID	Comment
R-1	Each product copy is represented by an L0 object.
R-2	The constraint `noInstances` in `NonIdentifiableProduct` prevents the creation of bulk products at L0.
R-3	The class `Product` defines common properties of all kinds of products.
R-4	represented by associations between descendants of `BulkProduct` and `ProductPackage`
R-5	The class `BulkPackage` is specialized from `NonIdentifiableProduct`.
R-6	ensured by descendants of `IdentityProduct` on L1, all product copies are instantiated from
R-7	ensured by the classes at L2 all classes representing product specifications are concretized from
R-8	satisfied by the intrinsic attribute `standardPrice` in `Product`
R-9	satisfied by the intrinsic constraint `reducedPriceSmaller` in `IdentityProduct`
R-10	satisfied by the intrinsic attribute `minPrice` in `Product` and the constraint `minimumPrice` in `NonIdentifiableProduct`
R-11	satisfied by the intrinsic attribute `minPrice` in `Product` and the constraint `minimumPrice` in `IdentityProduct`
R-12	addressed through the intrinsic attribute `currency` within the class `Product`
R-13	satified by the intrinsic constraints `CurrencyMatch1`, specified in `Product` and the constraint `CurrencyMatch0`, specified in `Product`
R-14	The operation `taxRate()`, specified in the L3 object `Product`, returns the respective tax rate for each tax type assigned (see below).
R-15	addressed by adding the enumeration type `Tax` that contains the values `NORMAL` and `REDUCED` and the attribute `taxType: Tax` in `Product`.
R-16	The intrinsic operation `priceAfterTax()`, separately specified in `IdentityProduct` and `NonIdentifiableProduct`, calculates the final price for L0 identity product objects and L1 bulk product objects.
R-17	satisfied by the two incarnations of the intrinsic operation `valueInStock` within the classes `IdentityProduct` and `NonIdentifiableProduct`
R-18	satisfied by the the operation `valueInStock` within the class `Product`
R-19	The intrinsic operation `priceAfterTax()`, separately specified in `IdentityProduct` and `NonIdentifiableProduct`, calculates the final price for L0 identity product objects and L1 bulk product objects.
R-20	demonstrated by the the operations `bulkValueInStock()`, `identityValueInStock()`, and `totalValueInStock()` defined in the class `WarehouseManager`
R-21	can be done through multiple concretizations of the classes `IdentityProduct`, `BulkProduct`, and `BulkPackage`
R-22	The intrinsic operation `priceAfterTax()`, separately specified in `IdentityProduct` and `NonIdentifiableProduct`, calculates the final price for L0 identity product objects and L1 bulk product objects.
R-23	satisfied by intrinsic attribute `introduced` in the class `Product`
R-24	addressed by the association `recommends`
R-25	The compatability of recommendation association is ensured through two modeling concepts. The association `mayRecommend` serves the specification of allowed recommendations. In addition, the constraint `properRecommendations` ensures that recommendations can be made only that were approved for the respective product specifcation types.
R-26	The constraint `noCyclicRecs` ensures that L1 descendants of `Product` cannot recommend themselves.

Fig. 4. `WarehouseManager` class on L1 and instance on L0

In general, the effort and risk related to changing a multi-level model widely depend on how well guidelines 1-3 had been followed. If these guidelines are satisfied, changing a model benefits from the tight dependencies between levels. If that is not the case, these dependencies turn into a serious disadvantage.

B. Further Aspects of Abstraction

The proposed solution clearly illustrates the benefit of the additional abstraction enabled by a multi-level modeling language like the FMML^X. A closer look, however, shows that there is still room for improvement. At first, this concerns additional languages constructs.

Our solution satisfies the requirement that a product specification may only recommend others to which it is entitled to R-25) by defining two associations and a constraint. However, since the specification of such a constraint is not trivial and this kind of dependency between two associations – where the dependent association is restricted to objects the classes of which are linked through instantiations of the independent association – is common, we consider adding a specific concept to the FMML^X as it is already provided by the LML and Melanee [22]. In this case, the implementation would be built on the generic pattern all corresponding constraints are based on. The convenient application of this concept recommends enhancing the concrete syntax of the FMML^X – for example by using a directed edge between the edges representing the respective associations. For a more comprehensive discussion of associations in multi-level models see [26].

It also inspired us to reconsider our view on potencies. To express at a higher level already that a class of a certain kind at a lower level must not be instantiated, the FMML^X requires the specification of a constraint, while this would not be required for the use of potencies.

Two further extensions of the language are clearly more demanding. At first, they concern concepts that enable abstractions of associations, e.g., by allowing for the specification of association (meta) types including the deferred instantiation of cardinalities. In addition, concepts that support dynamic abstraction would not only be extremely beneficial for modeling processes, but also for the specification of operations within classes. In these cases, one often sees commonalities but lacks the concepts to express these concisely.

C. Comprehensibility

As far as the comprehensibility of the model in general, and specifically of modeling constructs such as operations and constraints are concerned, we are reluctant to offer an

assessment. Even though there is still room for improvements, we are fairly satisfied with the FMML^{X} and XModeler^{ML}. But our view is certainly biased. While it is sometimes argued that multi-level models are difficult to understand, because people tend to struggle with abstraction, our extensive work with the construction of corresponding models and languages, as well as the experience we gathered with teaching MLM, indicate that the concepts represented in multi-level models are often perceived as more natural than the workarounds required in the traditional modeling paradigm. We think that the model presented here confirms this observation. Nevertheless, the specification of operations and constraints with the XOCL require some programming knowledge, as well as knowledge of the OCL.

VIII. Conclusion

The MULTI 2023 Warehouse Challenge proved to be a useful test case for us. It is suited to demonstrate the expressive power of the FMML^{X} and the utility of a development and execution environment like the XModeler^{ML}. At the same time, it served us to reconsider a few design decisions previously made with the specification of the FMML^{X}, which will likely lead to two specific extensions of the language.

Due to the nature of the MULTI challenge, only a small range of products was accounted for. While respective solutions should be suited to convincingly show the specific advantage of multi-level language architectures, models that cover a wide variety of products would allow for more impressively demonstrating the power of multi-level modeling and corresponding tools. While such a project would likely exceed the capabilities of single research groups, it may be an inspiring subject of an "open model" [15] project carried out by the MLM community.

References

[1] Colin Atkinson, Bastian Kennel, and Björn Goß. The Level-Agnostic Modeling Language. In Brian Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering*, pages 266–275, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Colin Atkinson and Thomas Kühne. Processes and Products in a MultiLevel Metamodeling Architecture. *International Journal of Software Engineering and Knowledge Engineering*, 11(06):761–783, 2001.

[3] Colin Atkinson and Thomas Kühne. Rearchitecting the UML infrastructure. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 12(4):290–321, 2002.

[4] Victorio A Carvalho, João Paulo A Almeida, Claudenir M Fonseca, and Giancarlo Guizzardi. Extending the Foundations of Ontology-based Conceptual Modeling with a Multi-Level Theory. In *Conceptual Modeling: 34th International Conference, ER 2015, Stockholm, Sweden, October 19-22, 2015, Proceedings 34*, pages 119–133. Springer, 2015.

[5] Tony Clark, Paul Sammut, and James Willans. Applied Metamodelling; A Foundation for Language Driven Development, 2nd ed. Sheffield: Ceteva, 2008. https://arxiv.org/pdf/1505.00149.pdf.

[6] Tony Clark, Paul Sammut, and James Willans. Developing Languages and Applications with XMF. Sheffield: Ceteva, 2008. https://arxiv.org/pdf/1506.03363.pdf.

[7] Tony Clark and James Willans. Software language engineering with XMF and XModeler. In *Formal and practical aspects of domain-specific languages: recent developments*, pages 311–340. IGI Global, 2013.

[8] Juan de Lara and Esther Guerra. Deep Meta-modelling with MetaDepth. In Jan Vitek, editor, *Objects, Models, Components, Patterns*, pages 1–20, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[9] Ulrich Frank. Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. *Domain engineering: Product lines, languages, and conceptual models*, pages 133–157, 2013.

[10] Ulrich Frank. Multilevel Modeling: Toward a New Paradigm of Conceptual Modeling and Information Systems Design. *Business & Information Systems Engineering*, 6(6):319–337, 2014.

[11] Ulrich Frank. Designing Models and Systems to Support IT Management: A Case for Multilevel Modeling. In *MULTI@ MoDELS*, pages 3–24, 2016.

[12] Ulrich Frank. Flexible Multi-Level Modelling and Execution Language (FMML^{x}): Version 2.0: Analysis of Requirements and Technical Terminology. Technical Report 66, ICB-Research Report, 2018.

[13] Ulrich Frank. Prolegomena of a Multi-Level Modeling Method Illustrated with the FMML^{x}. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 521–530. IEEE, 2021.

[14] Ulrich Frank and Tony Clark. Language Engineering for Multi-Level Modeling (LE4MM): A long-term Project to Promote the Integrated Development of Languages, Models and Code. In Jaime Font, Lorena Arcega, José Fabián Reyes Román, and Giovanni Giachetti, editors, *Proceedings of the Research Projects Exhibition Papers Presented at the 35th International Conference on Advanced Information Systems Engineering (CAiSE 2023), Zaragoza, Spain, June 12-16, 2023*, volume 3413 of *CEUR Workshop Proceedings*, pages 97–104. CEUR-WS.org, 2023.

[15] Ulrich Frank and Stefan Strecker. Open Reference Models – Community-driven Collaboration to Promote Development and Dissemination of Reference Models. *Enterprise Modelling and Information Systems Architectures*, 2(2):32–41, 2007.

[16] Ulrich Frank and Daniel Töpel. Contingent Level Classes: Motivation, Conceptualization, Modeling Guidelines, and Implications for Model Management. In Esther Guerra and Ludovico Iovino, editors, *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 622–631, New York, NY, USA, 2020.

[17] Manfred Jeusfeld and Christoph Quix. Meta modeling with ConceptBase. In *Proceedings of the 1st International Workshop on Meta Modeling and Corresponding Tools (WoMM 2005)*. University of Essen, 2005.

[18] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design Guidelines for Domain Specific Languages. *arXiv preprint arXiv:1409.2378*, 2014.

[19] Thomas Kühne. Tiefe Charakterisierung. In *Proceedings of Modellierung 2004*, pages 121–133, 2004.

[20] Thomas Kühne and Daniel Schreiber. Can Programming be Liberated from the Two-Level Style? Multi-Level Programming with DeepJava. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems, languages and applications*, pages 229–244, 2007.

[21] Thomas Kühne and Manfred Jeusfeld. The MULTI Warehouse Challenge. https://jku-win-dke.github.io/MULTI2023/files/MULTI%20Warehouse%20Challenge.pdf. Accessed: 2023-05-21.

[22] Arne Lange and Colin Atkinson. Multi-level modeling with MELANEE. In *MoDELS (Workshops)*, pages 653–662, 2018.

[23] Fernando Macias Gomez de Villar, Adrian Rutle, and Volker Stolz. MultEcore: Combining The Best of Fixed-Level and Multilevel Metamodelling. In *MULTI 2016: Proceedings of the 3rd International Workshop on Multi-Level Modelling co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2016)*, pages 66–75, 2016.

[24] Bernd Neumayr, Katharina Grün, and Michael Schrefl. Multi-Level Domain Modeling with M-Objects and M-Relationships. In *Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling-Volume 96*, pages 107–116. Citeseer, 2009.

[25] Zoltan Theisz and Gergely Mezei. An Algebraic Instantiation Technique Illustrated by Multilevel Design Patterns. In *MULTI@MoDELS*, pages 53–62, 2015.

[26] Daniel Töpel. Associations in Multi-Level-Modelling: Motivation, Conceptualization, Modelling Guidelines, and Implications for Model Management. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 502–510. IEEE, 2021.