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

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—In this paper, we present our contribution to this year's MULTI challenge concerned with the modeling of physical products in a warehouse. Based on a thorough analysis and interpretation 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 and can be used, e.g., to compute the present value of products in stock. 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, meta modeling, conceptual modeling, warehouse challenge**

I. Introduction

Research on multi-level modeling (MLM) has produced a variety of multi-level modeling languages and respective modeling tools (e.g., [3]–[10]). 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 comparison of languages and tools recommends considering use cases that are suited to illustrate specific differences. This aspiration has lead to the inception of the MULTI challenges, held as part of the annual workshop on multi-level modeling. 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 [11].

Products represent an especially suited subject for MLM because they exist in a considerable variety, which to manage 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., [4], [5], [12], [13], [19]).

In this paper, we present and discuss a solution to this year's challenge. To that end, we introduce parts of a multi-level modeling method we applied to develop the solution. 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). 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 as well as implications of the challenge for the advancement of the FMML^{X} and the XModeler^{ML} (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. Every class is an object at the same time, hence, it has state, may execute operations and is 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., [19]) to express this duality for two reasons. First, it is probably not well suited for conceptual modeling which, among other things, aims at bridging the gap between software systems and domain languages. Second, 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 [22]. 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 objects resulting from *concretizing* a class over multiple levels *descendants*, and the set of all descendants the *concretization tree* of that class.

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

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) that are required for its bottom-up definition. 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.

Associations may be defined between two classes at different levels. However, generalization/specialization relationships can be defined only between classes at the same level. Operations and constraints are defined with the executable object constraint language (XOCL)[2]([14], [15]). The FMML^{X} comprises a default notation that follows the UML style and, among other things, uses colors to distinguish levels. To customized the design of multi-level DSMLs, the default notation can be replaced by a domain-specific variant (see below).

The FMML^{X} is specified as a monotonic extension of "XCore", the meta-model of the executable meta-modeling facility (XMF) [14]. Since it implements a "golden braid metamodel architecture" [15, 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`. 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. 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 ("eXecutable Metamodelling Facility", [15]). It implements the FMML^{X} and allows the execution of FMML^{X} models. 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. An extension of the XModeler^{ML} supports the design of customized GUIs through the transparent integration of an external GUI builder.

The diagram shown in Fig. 1 and Fig. 4 illustrate the concrete syntax of the FMML^{X} as well as its default notation. More detailed descriptions of the FMML^{X}, its metamodel, XCore and XMF are available in [5] and [14]–[18].

III. Multi-Level Modeling Guidelines

Modeling methods suggested for the use of traditional language with one classification level only, or for the design of traditional DSMLs (e.g., [23], [24]) provide useful general modeling principles such as "separate invariant from variable parts of a system", but are not sufficient to guide the appropriate use of the additional concepts provided by multi-level languages. We followed the multi-level modeling guidelines proposed in [25] as an orientation for a methodical development of the presented solution. Table I gives an overview of selected design principles that are part of the guidelines.

The selection is based on their respective relevance for the challenge at hand. Apparently, the design principles do not work as a cookbook. Instead, each principle recommends a thorough analysis of the respective domain. Nevertheless, due to the scope and focus of our paper, we detain from an explanation of these guidelines and refer instead to the source. When we refer to a guidelines during the presentation of the solution, we provide a short rationale.

TABLE I
Multi-Level Modeling Guidelines from [25] (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

Different from previous challenges, this year's challenge [11] does not provide a list of enumerated requirements, but a description of the warehouse domain together with a glossary. In order to facilitate convenient references to the requirements, we analyzed the description in order to extract requirements and assign a unique identifier to each one. 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 a *d*, 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 products, exemplars (copies) of which 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* (**??**).

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

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

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 which 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 (**??**).

(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 cyclic recommendations should be excluded, we added a respective requirement (**??**).

(4) *Warehouse Management*. According to the challenge, the warehouse "needs to keep track 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 data when a product was sold. Over time, prices are likely to change. Therefore, one would also have to store the corresponding history. Finally, it would not correspond to the idea of managing a a warehouse, if one represented items that are actually no longer part of the warehouse. 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 product it stores, both at the product specification level (**??**) 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 in packs that demands for recording both the amount of unpacked bulk products and the amount of bulk product packs (R-19). (i)

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).

(4) *Focus on Particular Product Specifications*. Among other things, the warehouse includes DVDs and books. According to the challenge, a book copy is an instance of a book specification that containts bibliographic data such as title and name of author, sales price, etc. Similarly, a DVD containing a movie is regarded as an instance of a DVD type that includes a description of the represented movie. We decided not to follow the conceptualization suggested by the challenge explicitly. Instead, we distinguish between an *intellectual artefact*, such as as monograph or a movie and its representation on some kind of medium. An intellectual artefact is unique and cannot be copied. It can be *represented* by different kinds of media. A monograph can be represented as a traditional book or on a digital medium. A movie may be represented on some kind of data medium or as stream.

Doing without such a distinction would result in redundancy (e.g., the same monograph would be represented both a as a book type and as a DVD type). It would also violate guideline G-5: a particular monograph, e.g., is not a class that can be instantiated. As a concequence, the description of particular books and movies in the challenge requires needs to be reconstructed. For example: "(3) Moby Dick is a Book Spec ... MB copy 1 is a copy of Moby Dick" would translate to "The book type with the ISBN classifies books which represent the monograph Moby Dick". Note that this terminological distinction does not change the essence of the requirement in the challenge.

We refrain from explicitly accounting for further product examples given in the Challenge. We hope that the solution will be clear about how they are represented.

V. Model Presentation

The following presentation is divided in 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 kinds *identity products* (subsection V-C) and of bulk products (subsection V-D). Finally, we present a general warehouse management class that allows

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.	*a*

for multiple kinds of warehouse analysis (subsection V-B). Note that the description does not explicitly account for every requirement. In cases, where the satisfaction of a requirement is obvious, we sometimes leave it with its description in Table III. Also, with respect to space limitations, we cannot present all constraints and operations.

In addition to this presentation, the complete, executable model as well as a screencasts that demonstrates its use are also available at https://le4mm.org/multi-23/. The XModeler^{ML} can be downloaded from the webpages of our project LE4MM [26].

A. Generic Domain Knowledge

The generic domain knowledge is represented by four classes on L3 (see Fig. 1). `Product` specifies the properties common to all kinds of products (R-3). The classes `IdentityProduct` and `BulkProduct` are specialized from `Product` and represent generic knowledge that distinguishes bulk products from identity products. While it might, at first sight, appear to regard `IdentityProduct` and `BulkProduct` 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 guideline G-6.

Following the terminology of the case description, 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 either `BulkProduct` or `IdentityProduct` (R-21). Note that this addition and removal of product specification types is, however, restricted to the available classes on L3. Any future changes that might occur concerning, e.g., the composition of products or bulk products that become identity products, are not possible without adjustments. We discuss this aspect in more detail in the Discussion section (see section VII).

In particular, 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 represented in different currencies.

The attribute `currency` of type `Currency` addresses requirement R-12. This allows each L2 descendant of `Product`, which corresponds to a product specification type, to specify a different currency. The attribute `standardPrice` in `Product` is instantiated on L1 and thus enables each product specification to define a different SSP (R-8). The constraint `CurrencyMatch1` in `Product` ensurs 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
```

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 constraint `reducedPriceSmaller` defined with the same class ensures that reduced price defined on L0 is lower than the SSP assigned on L1 (**??**):

```
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 float value of a `MonetaryValue` value is retrieved via the `getAmount()` function. The `getAmountIn()` function 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 (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
```

Note that the classes defined as the context of these constraints serve as an abstraction of the specific classes or objects the constraints apply to at lower levels.

The intrinsic attribute `inStock` defined with `BulkProduct` 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 **??** and **??**. `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`, because they are products and we assumed that the warehouse does not keep track of single packs. 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 potentially different products, and, as such, might define different values for attributes like `introduced`, `taxType`, or `standardPrice`.

Note that this conceptualization does not reflect the knowledge that a pack contains bulk products (which would include other packs). While this is in some contrast to guideline V-A, it does not clearly violate the guideline, since at this level we cannot tell much about possible kinds of containment and their properties, especially cardinalities.

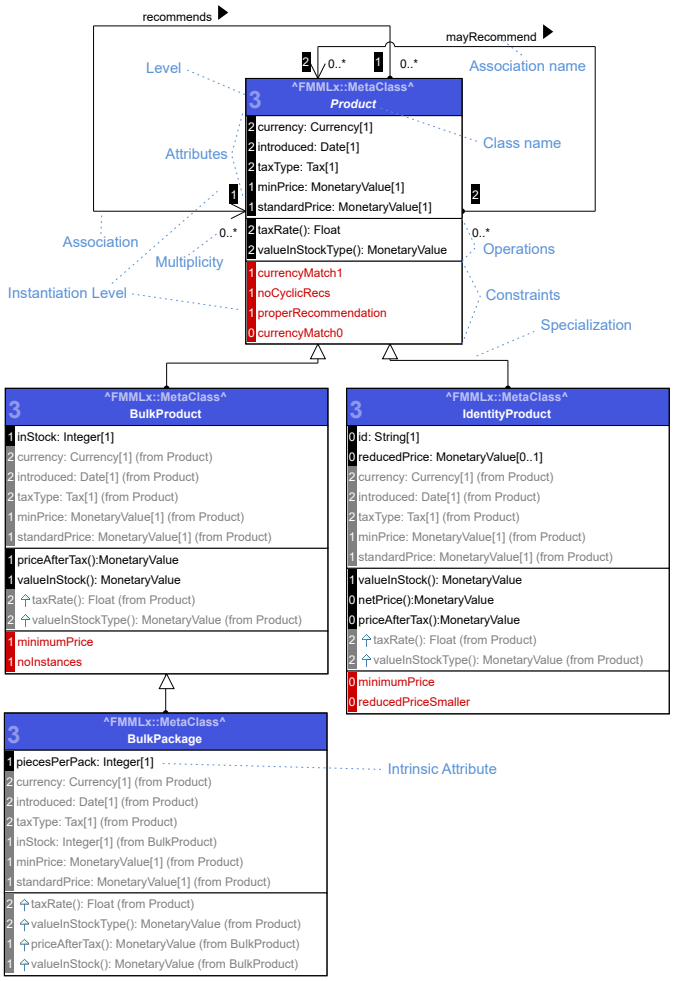

Fig. 1. `Product` class with specializations on L3

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.

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 `BulkProduct`, 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 that, the final price is calculated by the operation `priceAfterTax()`.

The accumulated value of product 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 `identityValueInStock()` based on the `priceAfterTax()` value returned by the L0 product copies of a product specification (see below). For L1 descendants of `BulkProduct`, the sum is calculated based on the `standardPrice` value on L1 by the intrinsic operation `bulkValueInStockBulkProduct`.

```
Context IdentityProduct L1
@Operation identityValueInStock():Auxiliary::MonetaryValue
  let sum = Auxiliary::MonetaryValue(0.0,Auxiliary::eur)
  in @For p in Challenge23::IdentityProduct.
     allMetaInstances()→select(o |
       o.level = 0) do
       sum := sum.add(p.priceAfterTax())
     end; sum
  end
end
```

Computing the accumulated value of product specification types (cf. R-18), however, does not require to distinguish between identity and bulk products. 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
```

The intrinsic association `recommends` that is to link descendants of `Product` at L1 serves to fulfill requirement R-24. The associations alone, however, is not satisfactory, since only products of a certain kind may be involved in recommendations (cf. R-25). To address R-25, 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 may not be recommended"
end
```

The constraint `noCyclicRecs` in `Producto` serves to further prohibit cyclic recommendations (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. According to **??**, no exemplars of bulk products may exist in the warehouse. The constraint `noInstances` within `BulkProduct` is to make sure that no bulk products must exist at L0.

B. Warehouse Management Class

While objecs on L2 that represent product specification types can iterate all their respective concretization tree, no class could iterate all product objects independent whether they bulk or identity product. The iteration of all product objects (cf. R-20) is realized through a separate `WarehouseManager` class (see Fig. 2). 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 `BulkProduct` 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`.

```
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 that fulfills R-20:

```
@Operation totalValueInStock():Auxiliary::MonetaryValue
  self.identityValueInStock().add(self.bulkValueInStock())
end
```

C. 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

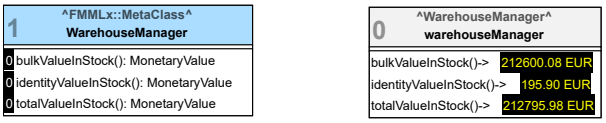

Fig. 2. `WarehouseManager` class on L1 and instance on L0

product specification types on L2 that, in turn, can be used to concretize product specifications on L1 that can be used to instantiate product copies on L0. In the following, we will only describe a few exemplary objects that concern this level of the challenge. The model includes more and can be easily populated with further instances.

Fig. 3 shows the identity product specification type `DVD` and its descendants on L1 and L0. It illustrated the distinction of intellectual artifacts from their representation. We added a few properties, like `length` for `Movie`, to underline this distinction. Note that the current version of the XModeler^{ML} requires names of associations to be unique within a model. Therefore, we used the designator `hasContent` instead of `represents`, which would have been more consistent. Removing this restriction is subject of a current revision of the XModeler^{ML}.

The object `DVD`, which represents a product specification type provides reference values to attributes defined in `Product` that have an instantiation level of 2. For example, `DVD` provides a `currency` value (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 `SpO1` (cf. R-1).

Operations that are executed by these objects return the respective value as defined in the `Product` or `IdentityProduct` class. The object `SpO1` returns a net price of 19.95 USD because no reduced price is provided (`reducedPrice=null`) and this corresponds to the SSP from its product specification `SpaceOdysee`.

D. Bulk Products and Bulk Packages

An example bulk product specification type from the case description is shown in Fig. 4. `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 packaging types – or none at all. We also added some battery-specific attributes to `BatteryCell` like `voltage` and `recharchable: Boolean`. `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. Both L1 objects therefore define an SSP, which is `1.50 NZD` for `EnergeticPlus` and `12.00 NZD` for `EnergeticPlus_Pack10`. The SSP of `EnergeticPlus` thereby corresponds to the price value of a single battery cell.

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. On the one hand, they concern the power of the language concepts used, and on the other hand, the effects of possible changes in the future.

A. Need for Potencies?

According to R-2, particular instances of bulk products must not be represented. We accounted for this requirement by modeling bulk product types as classes at L1. Since a class at L1 would allow for creating instances at L0, we added the constraint `noInstances`. Even though the definition of this constraint does not require considerable effort, it would be more convenient and safer to implement it as an additional language concept.

This could be achieved by extending the FMML^{X} with potencies. It would then be sufficient to define a potency of 2 with the class `BulkProduct` in order to prevent instances at L0. So far, the FMML^{X} does without potencies, because we assume that the meaning of a class chiefly depends on the level it conceptually belongs to. At the same time, the potency of a class can in most cases be derived from its level, which makes it widely redundant. However, the example of bulk products which are not accounted for on the level of particulars indicates that potency as an additional, though in many cases redundant concept, may be useful.

While we never experienced the need for potency as an additional concept in our previous work, we now realize that potencies might be a useful extension to the FMML^{X}. Therefore, we will analyse the utility of using potencies, also for the instantiation of properties, in our future work.

Alternatively, one could do without both, a constraint or a potency, by defining classes at L1 as abstract or by introducing an additional language concept, one may call “collective class” that would also prevent its possible subclasses from being instantiated. Such an approach would allow for instances of a bulk class at a later point in time (see C below).

B. Dependencies between Associations

To satisfy the requirements related to product recommendations (R-24, R-25), we used the two assocations `recommends` and `mayRecommend` as well as an additional 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} (similar to the one already available, e.g., in the LML (Melanee)). In this case,

Fig. 3. The descendants of `IndentityProduct`

Fig. 4. L2 and L1 descendants of `BulkPackage` and `BulkProduct`

the implementation would be built on the generic pattern all corresponding constraints are based on. Specific constraints differ only by the names of the access operations and the number of `.of()` operations, depending on the level distance. As such they could easily be generated on demand or a generic built-in constraint could be used for the concept. 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 associ-

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 `BulkProduct` 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 `BulkProduct`.
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 `BulkProduct`
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 `BulkProduct`, 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 `BulkProduct`
R-18	satisfied by the the operation `valueInStock` within the class `Product`
R-19	The intrinsic operation `priceAfterTax()`, separately specified in `IdentityProduct` and `BulkProduct`, 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` and `BulkProduct`
R-22	The intrinsic operation `priceAfterTax()`, separately specified in `IdentityProduct` and `BulkProduct`, 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.

ations in multi-level models see [29].

C. Possible Future Changes

While the design of the solution we present in this paper is solely aimed at fulfilling the requirements of the challenge, the design of conceptual models usually recommends accounting for possible future changes in order to improve a system's adaptability and, hence, sustainability. 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 like 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 assigned to them.

Second, related to requirement R-23, adding new product types may not only require additional classes at the level of `BulkProduct` and `IdentityProduct`, e.g., for perishable food that needs to be refrigerated or for financial products. Furthermore, other product types may require configuration, e.g., picking colours, materials, extra features, etc. This may lead to the need for additional levels, which, in turn, may require contingent level classes [21]. Ultimately, a versatile multi-level model for modeling products of any kind would consist of layers of DSMLs. At the top level, a generic product modeling language would provide all concepts shared by all kinds of products. At lower levels, more specific DSMLs would be added, e.g., for modeling cars, furniture, clothing, etc.

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" [32] project carried out by the MLM community.

REFERENCES

[1] C. Atkinson and T. Kühne, "The Essence of Multilevel Metamodeling," in UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5, 2001. Proceedings, 2001, pp. 19–33.

[2] OMG, OMG Unified Modeling Language, Version 2.5. [Online]. Available: https://www.omg.org/spec/UML/2.5

[3] C. Atkinson, B. Kennel, and B. Goß, "The Level-Agnostic Modeling Language," in Software Language Engineering: Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers, 2011, pp. 266–275.

[4] J. de Lara and E. Guerra, "Deep Meta-Modelling with MetaDepth," in Objects, Models, Components, Patterns: TOOLS 2010 Proceedings, J. Vitek, Ed., Berlin, Heidelberg: Springer, 2010, pp. 1–20.

[5] U. Frank, "Multilevel Modeling: Toward a New Paradigm of Conceptual Modeling and Information Systems Design," Business and Information Systems Engineering, vol. 6, no. 6, pp. 319–337, 2014.

[6] M. A. Jeusfeld and C. Quix, "Meta Modeling with ConceptBase," in Proceedings of the 1st International Workshop on Meta Modeling and Corresponding Tools (WoMM 2005), 2005.

[7] T. Kühne and D. Schreiber, "Can Programming be Liberated from the Two-Level Style? Multi-Level Programming with DeepJava," in OOPSLA '07: Companion to the 22nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion, Quebec, Canada, 2007, pp. 229–244.

[8] F. Macías, A. Rutle, and V. 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), Saint-Malo, France, 2016, pp. 66–75.

[9] V. A. Carvalho, J. P. A. Almeida, C. M. Fonseca, and G. 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, 2015, pp. 119–133.

[10] Z. Theisz and G. Mezei, "An Algebraic Instantiation Technique Illustrated by Multilevel Design Patterns," in MULTI 2015: Proceedings of the 2nd International Workshop on Multi-Level Modelling co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2015), Ottawa, Canada, 2015, pp. 53–62.

[11] T. Kühne, M. Jeusfeld, "The MULTI Warehouse Challenge," https://jku-win-dke.github.io/MULTI2023/files/MULTI%20Warehouse%20Challenge.pdf (accessed May 21 2023)

[12] C. Atkinson and T. Kühne, "Processes and Products in a Multi-Level Metamodeling Architecture," Intenational Journal of Software Engineering and Knowledge Engineering, vol. 11, no. 6, pp. 761–783, 2001.

[13] T. Kühne, "Tiefe Charackterisierung," in Proceedings of Modellierung 2004, 2004, pp. 121–133.

[14] T. Clark, P. Sammut, and J. Willans, Superlanguages: Developing Languages and Applications with XMF. Sheffield: Ceteva, 2008. Accessed: Jun. 27 2022. [Online]. Available: https://core.ac.uk/download/42487298.pdf

[15] T. Clark, P. Sammut, and J. Willians, Applied Metamodelling; A Foundation for Language Driven Development, 2nd ed. Sheffield: Ceteva, 2008. Accessed: Jun. 27 2022. [Online]. Available: https://eprints.mdx.ac.uk/id/eprint/6060

[16] T. Clark and J. Williams, "Software Language Engineering with XMF and XModeler," in Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, M. Mernik, Ed., Hershey, PA: IGI Global, 2013, pp. 311–340.

[17] U. Frank, "Designing Models and Systems to Support IT Management: A Case for Multilevel Modeling," 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), Saint-Malo, France, 2016, pp. 3–24.

[18] U. Frank, "The Flexible Multi-Level Modelling and Execution Language (FMMLx), Version 2.0: Analysis of Requirements and Technical Terminology," ICB Research Report 66, Universität Duisburg-Essen, Essen, 2018. [Online]. Available: https://doi.org/10.17185/duepublico/47506

[19] C. Atkinson and T. Kühne, "Rearchitecting the UML Infrastructure," ACM Transactions on Modeling and Computer Simulation, vol. 12, no. 4, pp. 290–321, 2002.

[20] C. Atkinson, R. Gerbig, and T. Kühne, "Opportunities and Challenges for Deep Constraint Languages," in 15th International Workshop on OCL and Textual Modeling: Tools and Textual Model Transformations, Workshop Proceedings, Ottawa, Canada, 2015, pp. 3–18.

[21] U. Frank and D. Töpel, "Contingent Level Classes: Motivation, Conceptualization, Modeling Guidelines, and Implications for Model Management," in MODELS '20: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2020.

[22] B. Neumayr, K. Grün, M. Schrefl, "Multi-Level Domain Modeling with M-Objects and M-Relationships," in Proceedings of the 6th Asia-Pacific Conference on Conceptual Modeling (APCCM), 20009, pp. 107–116.

[23] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel, "Design Guidelines for Domain-Specific Languages," in DSM '09: Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling, 2009, pp. 7–13.

[24] U. Frank, "Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines," in Domain Engineering: Product Lines, Languages, and Conceptual Models, I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin, Eds., Berlin, Heidelberg: Springer, 2013, pp. 133–158.

[25] U. Frank, "Prolegonema 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), 2021, pp. 521–530.

[26] U. Frank and T. Clark, "Language Engineering for Multi-Level Modeling (LE4MM): A Long-Term Project to Promote the Integrated Development of Languages, Models and Code," in Proceedings of the Research Projects Exhibition at the 35th International Conference on Advanced Information Systems Engineering (CAiSE 2023), 2023, pp. 97–104

[27] C. Atkinson and T. Kühne, "On Evaluating Multi-Level Modeling," in Proceedings of MODELS 2017 Satellite Event: Workshops (ModComp, ME, EXE, COMMitMDE, MRT, MULTI, GEMOC, MoDeVVa, MDETools, FlexMDE, MDEbug), Posters, Doctoral Symposium, Educator Symposium, ACM Student Research Competition, and Tools and Demonstrations co-located with ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems, Austin, TX, USA, 2017, pp. 274–277.

[28] T. Kühne and A. Lange, "Meaningful Metrics for Multi-Level Modelling," in MODELS '20: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2020.

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

[30] A. Lange and C. Atkinson, Multi-Level Modeling with MELANEE: A contribution to the MULTI 2018 Challenge, MODELS (Workshops), pp. 653–662, 2018

[31] B. Neumayr and C. G. Schuetza and M. Schref, "Dual Deep Modeling of Business Processes: A Contribution to the Multi-Level Process Challenge," Enterprise Modelling and Information Systems Architectures, Vol. 17, No. 7 (2022).

[32] U. Frank and S. Strecker, "Open Reference Models – Community-driven Collaboration to promote Development and Dissemination of Reference Models," Enterprise Modelling and Information Systems Architectures, Vol. 2, No. 2 (2007).