Multilevel Modelling with MultEcore
A Contribution to the MULTI 2017 Challenge

Fernando Macías*†, Adrian Rutle*, Volker Stolz*†
*Deparment of Computing, Mathematics and Physics
Western Norway University of Applied Sciences
P.O. Box 7030, 5020 Bergen, Norway
Email: {fmac,aru,vsto}@hvl.no
†Department of Informatics
University of Oslo
P.O. Box 1072 Blindern, 0316 Oslo, Norway

***Abstract*—In the context of MULTI 2017, and as a means of fostering discussion and test the limits of the paradigm, the Bicycle Challenge [1] was proposed to tackle the issue that multilevel modelling still lacks a strong conceptual basis, consensus and focus. This paper presents one solution to that challenge, i.e. creating a multilevel hierarchy that represents the domain of bicycles as products composed of different parts and with different features, starting from very abstract concepts (components with weight and basic parts) and ending with one particular model of bicycle with brand-specific parts, which in turn is instantiated by a specific bicycle. We analyse and "fix" the requirements, discuss them, and present our solution using the MultEcore tool.**

I. Introduction

The approach to deep metamodeling which we have used to solve this challenge is implemented in the MultEcore tool [2]. This tool combines the best from the two worlds: fixed-level metamodelling with its mature tool ecosystem, and multilevel modelling with an unlimited number of abstraction levels, potencies and multiple typing. Using our approach, model designers can seamlessly create a multilevel version of their hierarchies while still keeping all the advantages they get from fixed-level ones. MultEcore facilitates multilevel modelling without leaving the EMF (Eclipse Modeling Framework [3]) world, and hence allowing reuse of existing EMF tools and plugins. The tool (and the solution to this challenge) is available for download from http://prosjekt.hib.no/ict/multecore/.

A multilevel model hierarchy in MultEcore is defined as an ontological hierarchy of models with a fixed, common and generic topmost metamodel. In other words, the ontological hierarchy does not require a linguistic metamodel in order to be consistent, as opposed to the clabject-based proposals such as [4], [5]. Therefore, this hierarchy is similar to MOF, but does not restrict the number of new levels that the user can create. An overview of the conceptual framework behind MultEcore is displayed in Fig. 1. Note that since the ontological stack can grow downwards an arbitrary number of levels, these are indexed increasingly from the top. A detailed evaluation of our design choices and formalisations can be found in [2], [6].

The differentiating aspects of our conceptual framework are summarised as follows:

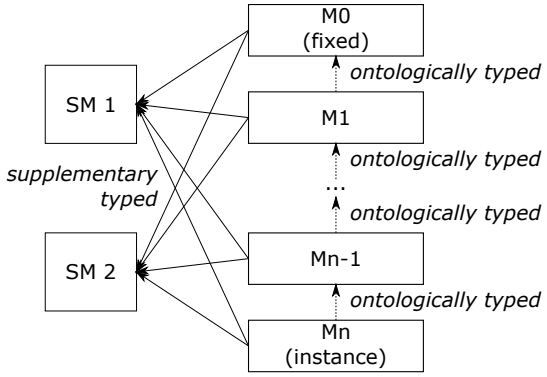

Figure 1: Overview of the multilevel conceptual framework

- A multilevel modelling stack that does not require linguistic metamodels, synthetic typing relations or "flattening" of the ontological stack.
- A realization of linguistic extension that differs from other approaches and allows for several, independent linguistic hierarchies (called supplementary hierarchies in our approach), orthogonal to the ontological stack.
- Looser linguistic typing: while every element in every model has an ontological type, it does not require a linguistic type for each *plugged* linguistic metamodel.
- An extension of the two-level cascading approach that aides the implementation of the framework as an extension of EMF which preserves full compatibility with its model representations and tools, i.e. we make use of EMF native APIs and formats, and keep the overload of the models as transparent as possible.

In the following sections we will show how we have designed the bicycle model in the MultEcore tool, and argue for our design decisions.

II. Case Analysis

The considerations we took to complete the case description are:

- Not specifying a value for attributes is interpreted as that attribute being optional.

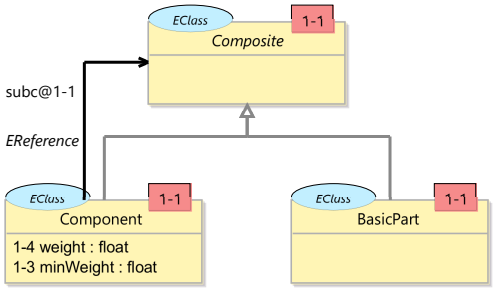

Figure 2: Level 1: Configuration

- All the bicycles are "well-formed". That is, the hierarchy does not allow bicycles with missing parts like a wheel.
- Unclear requirements that do not look suitable to be modelled are excluded.

The fact that the frame number for bicycles is unique can be specified higher up in the hierarchy but instantiated at the level of one particular bicycle. Moreover, the classifications defined in the description might be extended, adding even more intermediate levels of abstraction before reaching the actual, "final" instances, which represent individual bicycles. We have focused on the requirements and aspects which we have extracted from the challenge description (see next section).

III. Model Design

In this section, we present the multilevel hierarchy that addresses the challenge. We have one subsection per level, for the sake of clarity, starting from the top-most level 1. In level 0 we locate Ecore, which is not displayed. However, note that we will discuss the requirements sequentially as they appear in the challenge description [1], which causes changes in previously defined levels. These cases will be pointed out to avoid confusion.

About the visual representations, it is important to mention that the cardinality of the references is only displayed in case it is different from 0..*, which is the default one, and the most common and generic.

A. Level 1 - Configuration

The first configuration level resembles the traditional object-oriented Composite pattern [7]. Hence, this model exploits inheritance to achieve the pattern, as displayed in Fig. 2.

Apart from the classes themselves, the description mentions that components may have a weight and a minimum weight, included as attributes. The minimum weight attribute has potency 1–3, indicating that it can be instantiated directly in the level immediately below, two levels below or three levels below (levels L2, L3 and L4, respectively). This attempts to fulfil the requirement that the knowledge about the domain must be located as high as possible. In case we want to add more levels to our hierarchy, we may need to update this potency to a higher number. Also, the sentence "There is a difference between the type of a component and its instances" seems to clearly point out a separation between this level and the next one.

A hierarchy in MultEcore is tree-shaped and composed of models with upwards typing relations among them (see [6]). We have chosen the following concrete syntax in our tool. The type of a node is indicated as a blue ellipse, e.g. EClass is the type of Composite. The type of an arrow is written near the arrow in italic font type, e.g. EReference. The names of the nodes are used as labels in the class-like rectangles; italics font means the node is abstract. The potency of each node is written in a red rectangle on its top right corner. For arrows, it is written after the arrow name separated by the @ character. For attributes, the potency is written just before the attribute name. The potencies in MultEcore have a range, having the default "1–1", which means that they only can be instantiated directly at the next level below. This means that the tool does not restrict whether an instance of a node with potency 1–1 is reinstantiated or not. With the same reasoning, a node with potency 2–4 would mean that we can instantiate it directly at 2, 3 and 4 levels below, or a combination of those, or we may not do so since instantiation is optional. This realisation also keeps the potency of the instances independent of the potency of their types, and it could be seamlessly combined with the more traditional understanding of potency as depth (just by adding a third value), i.e. how many times instances, instances of instances, etc. can be created.

The tool also provides a hierarchy view in which all the models in a branch could be visualised as a modeling stack, with typing relations between model elements at different levels being visualised as dashed arrows.

B. Level 2 - Bicycle

The next level on the hierarchy contains the abstract description of a bicycle and its parts (see Fig. 3). Most of the requirements from the description are quite straightforward to model. The most relevant design decision is giving cardinality 0..1 to *purchasePrice* , since some instances of it do not specify it. It has a potency of 2–2 since there are still two levels to instantiate it below. Also, the fact that both wheels must have the same size has been solved by the attribute *wheelSize* in the class Bicycle hence forcing all wheels to be of the same size. This design decision can be replaced by the more complex (but arguably more semantically adequate) solution of duplicating the attribute in both wheels and creating a constraint that ensures that they have the same value. See subsection III-G for an example of a constraint.

One remarkable usage of potency in this level is that all the relations (except for *wheels*) are defined with potency 1–3 so that they do not need to be redefined in the intermediate levels while they can be directly instantiated in the lower levels. The other interesting use of our range-like potency is for both *color* attributes. We chose 1–4 for it since we believe it is a nice example of flexibility, so that a colour can be specified at the upper levels if that component is made with that colour, but we allow for the lower instances to redefine it, so that a particular bike can differ from its original colours.

The fact that frames have a unique serial number can be easily provided by exploiting Ecore's ID feature.

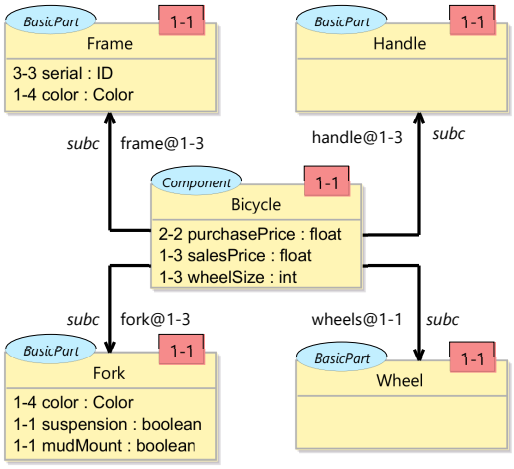

Figure 3: Level 2: Bicycle

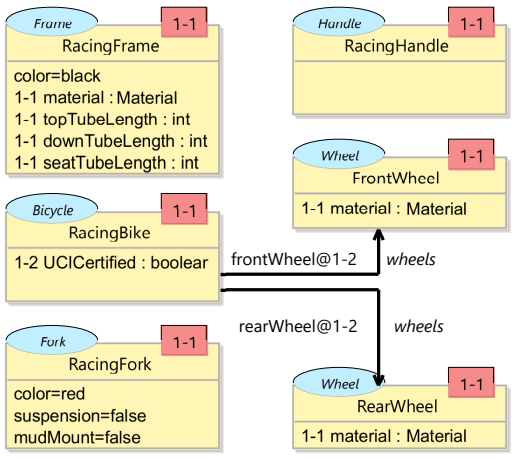

Figure 4: Level 3: Racing Bicycle

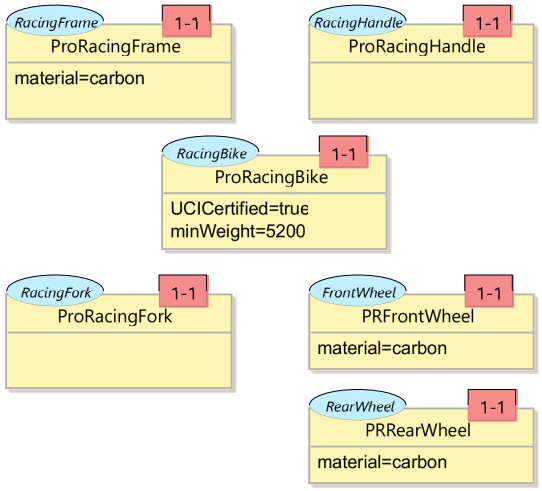

Figure 5: Level 4: Pro Racing Bicycle

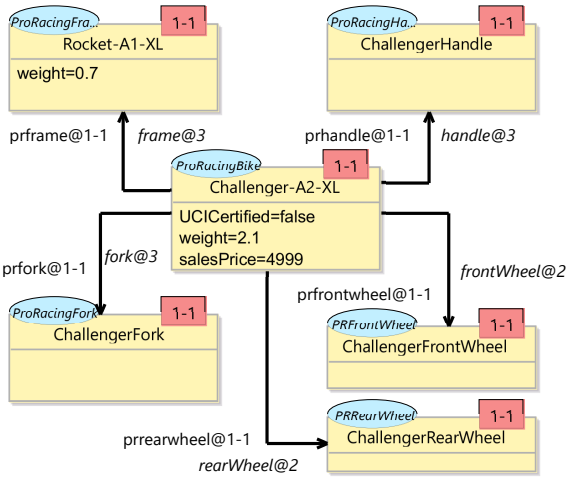

Figure 6: Level 5: Challenger A2-XL

C. Level 3 - Racing Bicycle

This level simply instantiates some attributes previously defined, and specifies new ones like the lengths of the different tubes for the frame (see Fig. 4). The fact that some bicycles are suitable to certain environments did not look suitable for explicitly modelling, since maintaining a list of them is cumbersome. At most, we consider that a simple string attribute with the description could be added.

D. Level 4 - Pro Racing Bicycle

This level is quite simple, and just instantiates the attributes previously defined (see Fig. 5). The most relevant feature is the restriction on the material of the wheels: "A carbon frame type allows for carbon or aluminium wheel types only". This requirement is addressed in section III-G.

E. Level 5 - Challenger A2-XL

The last level required by the description is a specific bicycle model, where we can see the instantiation of three attributes, *weight* (for both frame and bicycle) and *salesPrice*, defined at higher levels (see Fig. 6). Since *weight* is defined at level L1 together with *minWeight*, it is possible to also define a cross-level constraint where we forbid an actual weight to be less than the minimum weight. Specific instances of this model will instantiate the frame serial ID which will differentiate specific bicycles from each other.

F. Level 6 - My Challenger

The previous level defines a particular model of a bike, but probably more than one bike of that model is available. If the modeller wishes to represent this information, along with some specific differences between this bike an the rest of the Challenger A2-XL bikes, an instance of the previous model can be specified, like the one we depict in Fig. 7.

Here, the particular colours to which the bike was painted (or re-painted) can be specified, as displayed in *MyRocketFrame* and *MyFork*.

G. Constraints

Some of the features represented in the previous subsections could have been addressed by means of (multilevel) constraints,

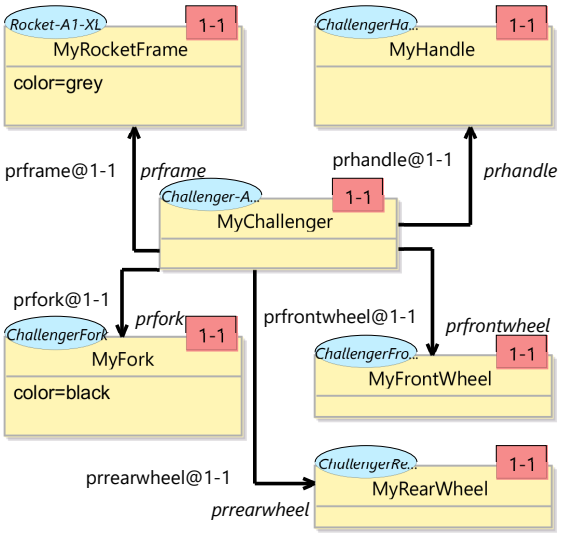

Figure 7: Level 6: My Challenger

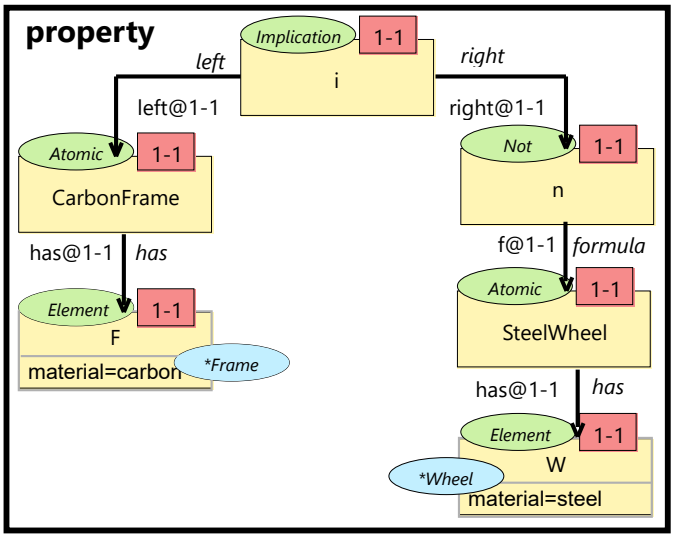

Figure 8: A multilevel constraint using a supplementary hierarchy

using a language like the one we describe in [6]. We chose, however, to create a hierarchy as simple and self-contained as possible. Hence, the only constraint we create is specified as an implication, in the following manner: A frame with the attribute *material=carbon* implies that either the front and rear wheels have also *material=carbon* or that they have *material=aluminium*. This constraint uses a supplementary hierarchy, as depicted in Fig. 1, in order to define double-typed elements that relate the atomic propositions of the language of boolean logic to actual elements in the ontological hierarchy (called application hierarchy in our approach). Fig. 8 displays the constraint, using the same notation as the previous diagrams, but with two different colours to distinguish between the application type (blue) and the supplementary type (green).

IV. Evaluation

The proposed multilevel modeling hierarchy ended up having up to 7 abstraction levels L0, . . . , L6, where the L0 level is the Ecore metamodel. The knowledge domain is at level L2, just below the generic component model at level L1.

The model at L2 can be used as a DSL, or as a starting point for a software system which could be used by bicycle retailers. This would imply that some of the potencies in L2, e.g. the ones on *purchasePrice*, *serial*, etc., would be updated so that these attributes could be instantiated at L3. Similarly, the model at level L3 can be used as a DSL, or as a starting point for a software system which could be used by racing bicycle retailers. An ordinary bicycle model, which is specified at level L3 as an instance of L2, would be in a sibling branch of the metamodel Racing Bicycle in the model hierarchy.

The Racing Bicycle specialised DSL or software could further be refined and specialised to define a Pro Racing Bicycle metamodel. This specialised DSL would disallow racing bicycle and pro racing bicycle retailers from defining ordinary bicycles. However, by changing the potency of the nodes at level L2 so that the upper bound is *, we could relax on this restriction, if this was desirable. Hence, although the refinement process has given rise to more specialised DSLs for special kinds of bicycle, e.g. pro racing bicycles, we could still choose to create a DSL which enables usage of types from upper levels than the Pro Racing Bicycle. That is, in our alternative solution with relaxed potencies, a specific ordinary bicycle could be specified at level L3, L4, L5 or L6 depending on which created DSL one would like to use.

In http://prosjekt.hib.no/ict/multecore/ we show how Sirius [8] could be used to create editors for a bicycle DSL given by the metamodel at level L2. Indeed, editors could be created for any of the models in the hierarchy. This also demonstrates the strength of our approach in not leaving the EMF world which makes it easy to create editors and other artefacts. Moreover, from the `.ecore` version of the models it is possible to use EMF's native code generation facilities and generate Java code, or write other templates to generate custom code.

In a few cases in this solution, we could have used generalisation (i.e. inheritance) instead of specialisation (i.e. typing). For example, the specialisation of the Racing Bicycle into Pro Racing Bicycle could be replaced with inheritance among the elements of both. However, we believe that the usage of typing in this scenario allows for a more flexible specification and separation of abstraction levels. As we argue in [6], our framework leaves this possibility open to the user.

In our solution, it is not required to have associations between different levels. In the case of cross-level constraints, we have used multilevel constraints as shown in Fig. 8. These constraints would be parsed and transformed to validators by customized code-generators so that the created DSLs can enforce the restrictions given by them.

We have identified three of the requirements given by the challenge description [1], namely the domain knowledge is

specified at the highest possible level, the models could be used as a foundation for a software system, and possibility to define cross-level constraints.

The main limitation of the approach lies in the fact that we cannot automatically propagate changes done to higher level models to lower level models. That is, if we change potencies or add/delete model elements, the lower level models which are depending on the potency or these elements might become invalid models. Addressing this challenge is part of a bigger development step in MultEcore which focuses on co-evolution and model repair.

V. Conclusions

In this paper, we have presented a solution to the Bicycle Challenge proposed at MULTI 2017 workshop. Our multilevel modeling hierarchy ended up having up to 7 abstraction levels where specific ordinary bicycles could be defined at the level L3, and with some potency relaxation also on L4, L5 and L6. However, specific pro racing bicycles could only be defined at level L5 since these are instances of a more specific or refined metamodel at L4. Our solution is based on the MultEcore tool and follows a conceptual framework which enables EMF with the potential of becoming a multilevel modelling framework. This facilitates usage of the rich ecosystem of EMF such as code generation and DSL editor creation.

References

[1] MULTI2017. (2017, Jul.) Bicycle Challenge description. [Online]. Available: https://www.wi-inf.uni-duisburg-essen.de/MULTI2017/#challenge

[2] F. Macías, A. Rutle, and V. Stolz, "MultEcore: Combining the best of fixed-level and multilevel metamodelling," in *MULTI*, ser. CEUR Workshop Proceedings, vol. 1722, 2016.

[3] Eclipse Modeling Framework, *Web site*. [Online]. Available: http://www.eclipse.org/modeling/emf

[4] C. Atkinson and R. Gerbig, "Flexible deep modeling with Melanee," in *Modellierung 2016*, ser. LNI, S. Betz and U. Reimer, Eds., vol. 255. Bonn: Gesellschaft für Informatik, 2016, pp. 117–122.

[5] J. de Lara and E. Guerra, "Deep meta-modelling with Metadepth," in *Objects, Models, Components, Patterns*, ser. LNCS, vol. 6141. Springer, Jul. 2010, pp. 1–20.

[6] F. Macías, A. Rutle, V. Stolz, R. Rodriguez-Echeverria, and U. Wolter, "Formalisation of flexible multilevel modelling," *Submitted, available at http://prosjekt.hib.no/ict/multecore/*, 2016.

[7] E. Gamma *et al.*, *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1994.

[8] The Eclipse Project, "Eclipse Sirius," Dec. 2016. [Online]. Available: http://www.eclipse.org/sirius/