Multilevel Modeling with MultEcore
A Contribution to the MULTI 2017 Challenge

Fernando Macías[1,2], Adrian Rutle[1], and Volker Stolz[1,2]

[1] Western Norway University of Applied Sciences
{fmac,aru,vsto}@hvl.no
[2] University of Oslo

Abstract In the context of MULTI 2017, and as a means of fostering discussion and test the limits of the paradigm, the Bicycle Challenge 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. We analyse and "fix" the requirements, discuss them, and present our solution using the MultEcore tool.

1 Introduction

The approach to deep metamodeling which we have used to solve this challenge is implemented in the MultEcore tool [3]. 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 linguistic extensions. 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 [1]) world, and hence allowing reuse of existing EMF tools and plugins. The tool (and the solution to this challenge) is available for download in `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. 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

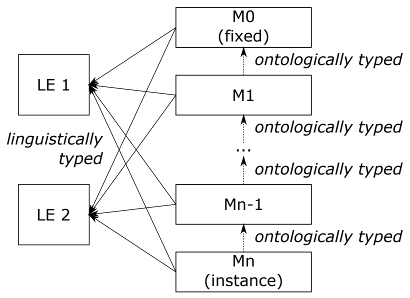

Figure 1: Overview of the multilevel conceptual framework

stack can grow downwards an arbitrary number of levels, these are indexed increasingly from the top. The differentiating aspects of our conceptual framework are summarised as follows:

- 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 metamodels 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.

2 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.
- 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.

Although some approaches are heavily based in the concept of depth for potency, in MultEcore we do not want to constrain the number of abstraction levels when designing the top ones. For example, 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. Similarly, the classifications defined in the description might be extended, adding even more intermediate levels of abstraction before reaching the actual, “final” instances. Hence, giving depth might in some cases act as a double-edged sword that makes it more difficult to allow the hierarchy to grow when faced with new requirements.

3 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 [5], 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.

3.1 Level 1 - Configuration

According to the challenge description, the most abstract level consists of components that can be composed of other components or basic parts. This resembles quite closely the traditional object-oriented Composite pattern [2]. Hence, this model exploits inheritance to achieve the pattern, as displayed in Fig. 2.

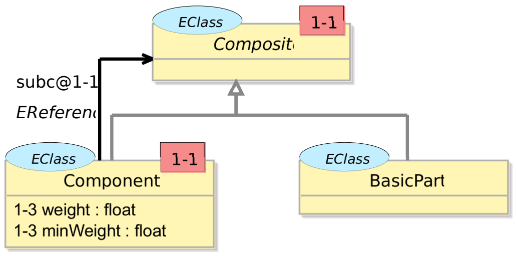

Figure 2: Level 1: Configuration

Apart from the classes themselves, the description mentions that components may have a weight, included as attribute. This 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 would 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.

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. 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.

3.2 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*, instead of using a constraint that would be more cumbersome. The fact that frames have a unique serial number can be easily provided by exploiting Ecore's ID feature.

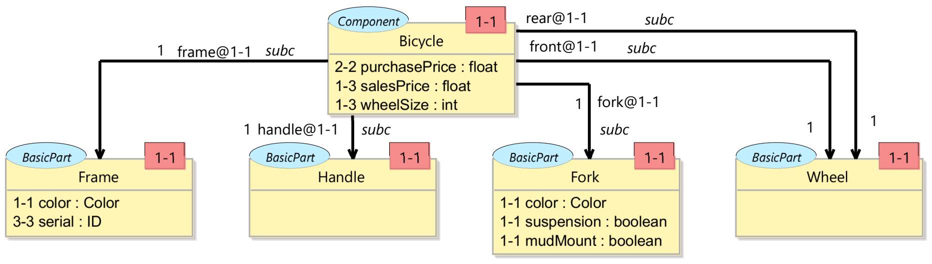

Figure 3: Level 2: Bicycle

3.3 Level 3 - Racing Bicycle

This level simply instantiates some attributes previously defined and specify 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.

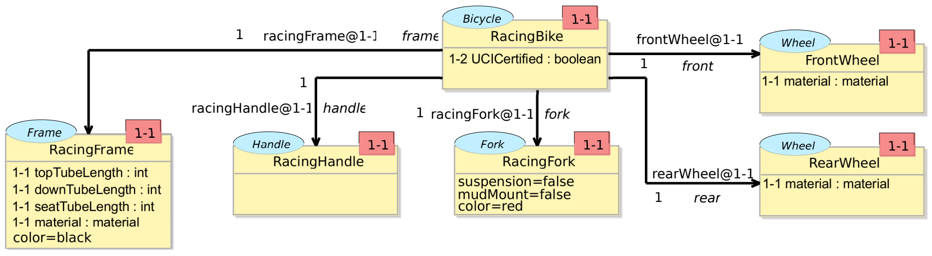

Figure 4: Level 3: Racing Bicycle

3.4 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 3.6.

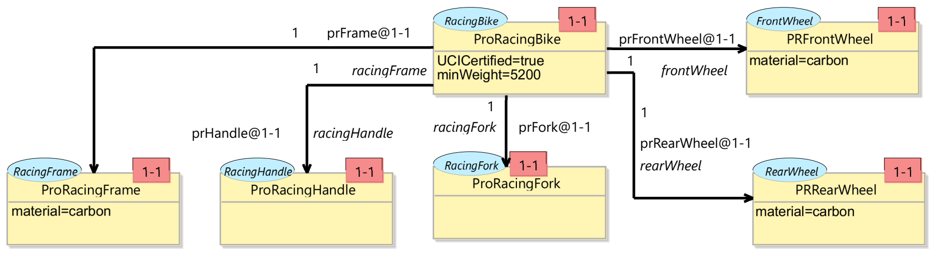

Figure 5: Level 4: Pro Racing Bicycle

3.5 Level 5 - Challenger A2-XL

The last level required by the description is a specific bicycle model, where we can see the instantiation of two attributes, *weight* and *salesPrice*, defined several levels above (see Fig. 6).

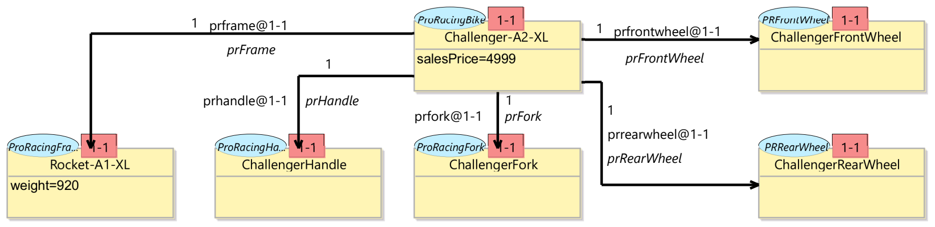

Figure 6: Level 5: Challenger A2-XL

3.6 Constraints

Some of the features represented in the previous subsections could have been addressed by means of (multilevel) constraints, using a language like the one we describe at [4]. 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*.

4 Evaluation

The proposed multilevel modeling hierarchy ended up having up to 6 abstraction levels L0, . . . , L5, 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. 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 an instance of L2 at level L3, would be in a sibling branch of the metamodel Racing Bicycle in the model hierarchy.

This Racing Bicycle specialised DSL or software could further be refined and specialised to define a Pro Racing Bicycle metamodel. This specialised DSLs 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 or L5, depending on which created DSL one would like to use.

In `http://prosjekt.hib.no/ict/multecore/` we show how Sirius [6] 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 approaches 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 our solution, it is not required to have associations between different levels. However, if it was necessary, we would define them as cross-level constraints.

5 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 6 abstraction levels where specific ordinary bicycles could be defined at the level L3, and with some potency relaxation also on L4 and L5. However, specific pro racing bicycles could only be defined at level L5. 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. Eclipse Modeling Framwork. *Web site*. http://www.eclipse.org/modeling/emf.
2. E. Gamma et al. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1994.
3. F. Macías, A. Rutle, and V. Stolz. MultEcore: Combining the best of fixed-level and multilevel metamodelling. In *MULTI*, volume 1722 of *CEUR Workshop Proceedings*, 2016.
4. 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.
5. MULTI2017. Bicycle Challenge description, July 2017.
6. The Eclipse Project. Eclipse Sirius, Dec. 2016.