Multi-level modeling with MELANEE

A contribution to the MULTI 2018 Challenge

Arne Lange, Colin Atkinson

University of Mannheim
Software Engineering Group
{atkinson, lange}@informatik.uni-mannheim.de

Abstract. This paper presents a solution to the MULTI 2018 Bicycle Challenge developed using the *MELANEE* deep modeling tool. The structure of the paper therefore follows the guidelines laid out in the Challenge description. After first outlining the case study and documenting which aspects are supported in the *MELANEE* solution, the paper presents a detailed description of the developed deep model. This is followed by a discussion of the strengths and weaknesses of the approach and a discussion of its benefits over traditional two-level modeling. The presented model covers all mandatory and optional aspects of the Challenge case study.

1 Introduction

The MULTI 2018 Bicycle Challenge was established by the organizers of MULTI 2018 as a vehicle for evaluating and comparing multi-level modeling approaches. This paper presents a solution to the Challenge developed using the *MELANEE* deep modeling environment [14]. This supports a style of multi-level modeling often referred to as deep modeling [6] since it is based on the use of (a) the deep instantiation mechanism (using potency) to represent ontological classification relationships [6] and (b) the tenet of strict modeling to define the different layers within a model [7].

For any modeling approach there are no hard and fast rules about what constitutes a good model as opposed to a bad model. In other words, models can be optimized for different purposes based on multiple criteria (e.g. minimality, readability, maintainability). The deep model presented in this paper, which satisfies all mandatory and optional requirements laid out in the Challenge, has been optimized to showcase *MELANEE's* features and thus does not claim to be the best model for a specific real-world modeling use case. Once the idiosyncrasies of *MELANEE's* multi-level modeling approach have been outlined in section 2, a complete overview of our solution to the Bicycle Challenge is presented in section 3. Section 4 then discusses the strengths and weakness of this deep model, before section 5 wraps up with some conclusions.

2 Multi-Level Modeling Approach

A full description of the case study at the heart of the MULTI 2018 Bicycle Challenge is available as an addendum to the MULTI 2018 call for papers [1] and is not replicated here. In this section we outline the distinguishing features of our solution built using the *MELANEE* multi-level modeling environment. As mentioned above, *MELANEE* supports a flavor of multi-level modeling often referred to as deep modeling. This means that ontological classification relationships are governed by potency and the modeling levels are defined according to the principles of strict modeling. In our approach,

therefore, we refer to the complete multi-level model encompassing all domain information as a "deep model" and each individual ontological level within this deep model as a "level" or a "model".

In *MELANEE*, deep models are represented using two (sub) languages - the so called Level-agnostic Modeling Language (LML) and a variant of OCL enhanced to be "aware of", and exploit, deepness. The LML contains three core constructs – "Entities", "Connections" and "Generalizations". Entities correspond to classes and/or objects in classic UML-style structural modeling and are depicted in a similar way, while connections correspond to association classes in the UML (although they are represented in a different notation). Connections can be navigable or non-navigable and can capture two forms of containment, composition and aggregation, using similar modifiers to the UML. Entities and Connections are Clabjects which have a potency indicating over how many levels they can be instantiated. The generalization relationship is used to represent inheritance using the unfilled-triangle notion of the UML to designate super-types [11].

MELANEE is a domain specific language workbench that supports a variety of formats and visualizations. It can be used to model in graphical, text-based, table-based and/or form-based formats to provide stakeholders with diverse viewpoints on a deep model and multiple ways to edit it. These visualizations can also be context aware, e.g. the background color of an entity can be dynamically determined via a deep OCL expression [11].

Our deep model solution to the Bicycle Challenge contains four ontological levels. Like the UML infrastructure, these are typically depicted in a vertical hierarchy with the most abstract (i.e. meta) level at the top and the most concrete (i.e. instance) level at the bottom. However, by convention *MELANEE's* levels are numbered in the reverse order to the UML's with the most abstract given the number 0, the second most abstract the number 1 and so on. *MELANEE* also allows arbitrary names to be given to levels to better characterize their content.

3 Model Design

The presented *MELANEE* model fully covers all mandatory and optional requirements defined in the Challenge description. However, two of the mandatory elements are renamed in our solution – the notion of `Configuration` is covered by `Product` (i.e. instances of `Product` are `Configurations`) and the notion of `Categories` is covered by `BicycleConfiguration` (i.e. subclasses of `BicycleConfiguration` are specific `Categories`).

3.1 Level-Spanning Domain Content

To understand the ontological levels themselves, it is necessary to understand the model information that spans them – so called "pan-level" information.

Linguistic (Meta) Models. The most fundamental pan-level models are the linguistic (meta) models of the LML and the deep OCL dialect used to define constraints. These are metamodels in the sense that they define languages, but are not strictly "meta" to anything since the content of the ontological levels exists at the linguistic level immediately below them. The LML linguistic model, which contains about a dozen classes, defines the core concepts of deep modeling mentioned in the previous section (e.g. clabjects, generalizations, attributes etc.) and has been designed to be as simple and minimal as possible whilst providing a UML-like modeling experience for modelers. The deep OCL variant supported by *MELANEE* is based on the OMG's OCL version 2.4. Its linguistic (meta)model has been enhanced to make it "aware" of (i.e. support) constraints over multiple ontological levels. In particular, its allows constraints to be applied to explicit ranges of ontological levels and to refer to linguistic as well as ontological attributes.

Pan-Level Enumeration Types. Most domains have domain specific data (i.e. value-only) types that are used at multiple ontological levels. In ***MELANEE*** these are defined within a deep model, but outside any specific ontological level so that in effect they span (i.e. are usable in) any level. For the Challenge two enumerations are defined – `Material` and `CyclistSize`. The `Material` enumeration, which is the type for the `material` attribute of racing frames, has three values `CARBON`, `ALUMINUM` and `STEEL`. The `CyclistSize` enumeration, which is the type for the `cyclistSize` attribute in the `Purpose` clabject, also has three values `TALLCYCLIST`, `MEDIUMCYCLIST` and `SMALLCYCLIST`.

Pan-Level Constraints. A powerful feature of the deep OCL dialect supported by ***MELANEE*** is that it allows level spanning constraints to be defined that control the way clabjects can be used within specific ontological level and/or in the whole deep model. This feature essentially supports the notion of reflexive constraints [10], since level spanning constraints can control the fundamental way the LML is used as exemplified by constraint *PAN-1* below.

```
context DeepModel
inv PAN-1: Clabject -> forAll(select(c|c.#getFeature()# -> select(f|f.#getDurability()#
> 0)) -> size() = self.getDirectInstances() -> select(c|c.#getFeature()#) -> size())
```

Although strict modeling requires the ontological type of a clabject to reside at the level immediately above it, it does not specifically require that all clabjects have an ontological type. Because ***MELANEE's*** orthogonal classification architecture gives every clabject a linguistic type (i.e. `Clabject`) as well as (possibly) an ontological type, it is perfectly possible to define ontologically-unclassified clabjects in a model (i.e. clabjects without an explicit, direct ontological type). This is important in practice since it allows the top (i.e. most abstract) ontological level to be populated with clabjects that are not ontologically classified, and thus avoids the endless repetition of levels that would be needed if every clabject had to be ontologically typed. It can also be useful to define ontologically unclassified clabjects at lower levels, giving rise to a specific style of deep modeling. The purpose of constraint *PAN-1* is to ensure that every clabject that has an ontological type has all, and only, the features (i.e. attributes and methods) required by the type. More specifically, it states that if a clabject has an ontological type, it must possess all features defined by that type with a durability greater than 0 and no more. Thus, only ontologically-untyped clabjects can introduce new features. The '#' symbol in the constraint is used to designate the linguistic dimension and invoke linguistic methods.

3.2 Product Level - O_0

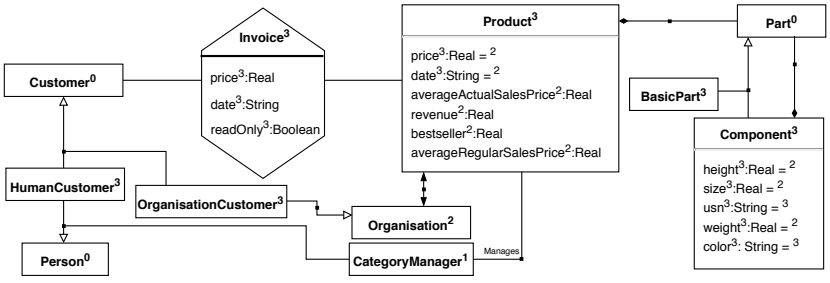

Fig. 1. Product Model (O_0)

Figure 1 shows the top, most abstract, ontological level of the deep model which captures the domain of selling products to customers. A `Product` is defined as a composite of `Components` and `BasicParts` using the composite pattern. `Products` can be certified by an `Organization`. An important feature of the deep model is that this top level is completely independent of the bicycle shop domain and thus can be instantiated for other domains.

The LML uses clabjects called connections to represent associations between entities, but they are more like UML association classes than plain associations since they can have attributes, super types and participate in connections themselves. Connections can be depicted in two ways in ***MELANEE*** models depending on the amount of information the modeler wishes to display. The connection between `Product` and `Customer`, called `Invoice`, is depicted in expanded form in Figure 1 to show its two attributes, while the connection between `CategoryManager` and `Product`, called `Manages`, is shown in collapsed form because it has no attributes.

The attributes of `Invoice` document the essential properties of sales transactions. The other kind of relationship appearing in Figure 1 is specialization which is depicted using the UML unfilled-triangle notation. The figure highlights the important point that the potencies of sub-clabjects need not be the same as those of their super-clabjects because potency is based on direct classification relationships (as opposed to indirect classification relationships arising from inheritance hierarchies).

The attributes `averageActualSalesPrice`, `averageRegularSalesPrice`, `revenue` and `bestseller` of the clabject `Product` are responsible for storing the optional sales information described in the section 2.2 of the Challenge description. Their values are defined by the following four derive constraints which are applied at specifically defined levels. Constraint $O_0 1$ derives the value of the `averageActualSalesPrice` for every instance of `Product` at O_1 and O_2. The context of the derivation is iterated over every instance of `Product` so that the value is derived for each individual instance. It therefore covers the first two derived properties of section 2.2 of the Challenge. The constraint $O_0 2$ defines the value of the `averageRegularSalesPrice` attribute in a similar manner as $O_0 1$. Constraint $O_0 3$ defines the value of the `revenue` attribute while $O_0 4$ determines the top-seller of each category and the top selling category.

```
context Product::averageActualSalesPrice:Real
derive O0 1: self.allInstances() -> select(c|c.#getPotency()# = 0) ->
select(c|c.Invoice.date.substring(7,10) = "2017") -> collectNested(Invoice.price)
-> sum() / self.allInstances() -> select(c|c.#getPotency()# = 0) -> size()

context Product::averageRegularSalesPrice:Real
derive O0 2: self.allInstances() -> select(c|c.#getPotency()# = 1)
-> select(date.substring(7,10) = "2017") -> collect(price) -> sum() /
self.allInstances() -> select(c|c.#getPotency()# = 0) -> size()

context Product::revenue:Real
derive O0 3: self.allInstances() -> select(c|c.#getPotency()# = 0) ->
select(c|c.Invoice.date.substring(7,10) = "2017") -> collectNested(Invoice.price)->sum()

context Product::bestseller:Boolean
derive O0 4: let topSeller:Boolean = false in
if Clabject -> select(c|c.isDeepKindOf(BicycleConfiguration) = true) ->
select(date.substring(7,10) = "2017") -> sortedBy(revenue) -> last() = self
then topSeller = true else topSeller = false endif
```

3.3 Bicycle Categories Level - O_1

Figure 2 shows the second level of the deep model, O_1 where the ontological instances of clabjects in O_0 reside. This level describes the structures of the different kinds of bicycle product categories as well as their different roles and stakeholders. The tenet of strictness requires all ontological instances of O_0 to reside at O_1, but does not require all elements in O_1 to have a direct ontological type.

For example, the clabject `ProfessionaleRacingFrame` has no ontological type. This is because it needs more attributes than a "normal" `Component`, so it inherits the normal component attributes from `Frame` (an instance of `Component`) and adds its own attributes relevant to professional racing frames. There are two connections needed between `BicycleConfiguration` and `Wheel`, one for the front wheel and one for the rear wheel. Every instance of `Product` is connected to a `Purpose`. Note that all the connections are depicted in collapsed form in Figure 2. Although this means their attributes cannot be seen, it is still possible to display their names in the style of UML associations.

UML-style multiplicity constraints are used to indicate that a `BicycleConfiguration` must have exactly one `Frame` and `Fork`. In addition, seven deep-OCL constraints are needed to fulfill the mandatory requirements of the Challenge. Constraints $O_1 1$ and $O_1 2$ are defined on `BicycleConfiguration`. Constraint $O_1 1$ ensures that the front wheel and the rear wheel have the same size. If a bike has a carbon frame it needs to have carbon or aluminum wheels as well. This is expressed in

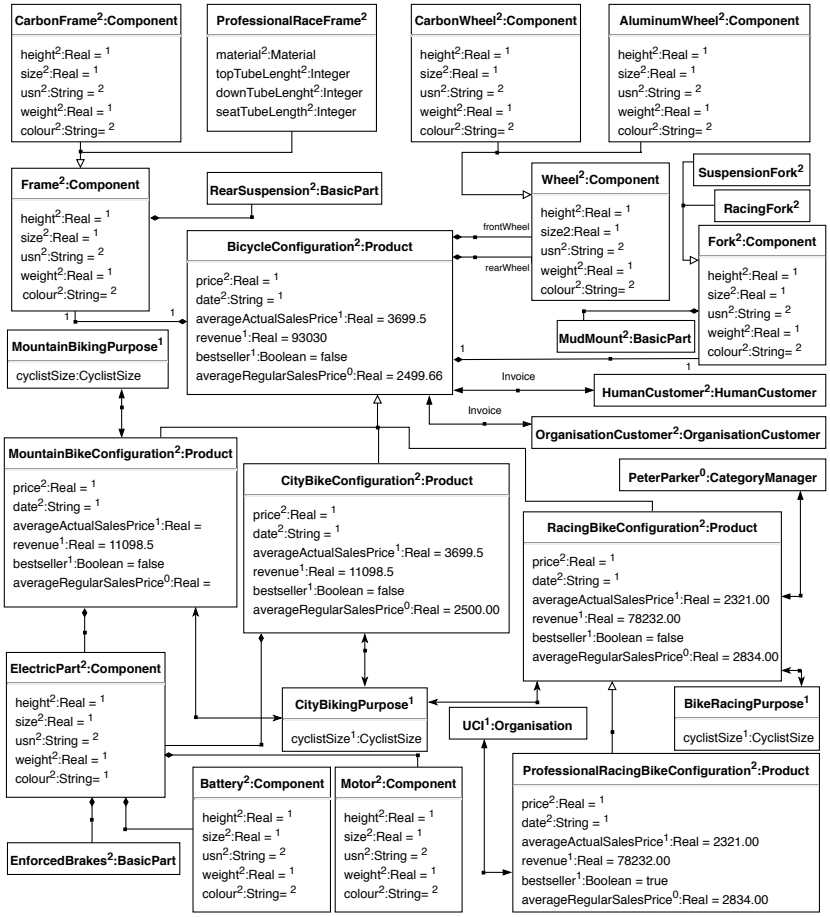

Fig. 2. Bicylce Categories Model (O_1)

constraint $O_1 2$. Constraints $O_1 3$ and $O_1 4$ are defined for `RacingBikeConfiguration`. Constraint $O_1 3$ states that every fork to which a `RacingBikeConfiguration` is connected must be an instance (direct or indirect) of `RacingFork` and every frame must be an instance (direct or indirect) of `ProfessionalRacingFrame` while constraint $O_1 4$ ensures no instance of `RacingBikeConfiguration` has a `MudMount` connected to it. Constraints $O_1 5$ and $O_1 6$ are defined for `ProfessionalRacingBikeConfiguration` and ensure the `weight` attribute is above the required minimum weight of that bike category and that the bike is certified by an organization. The last constraint, $O_1 7$, ensures that no `CityBikeConfiguration` is connected to a `RearSuspension`. Every invariant constraint displayed here is applied to all instances of each context at the O_2 and O_3 level.

```
context BicycleConfiguration
inv O1–wheelSize: self.frontWheel.size = self.rearWheel.size

context BicycleConfiguration
inv O1 2–carbonFrame: self.frame.isDeepKindOf(CarbonFrame) implies
(self.frontWheel.isDeepKindOf(CarbonWheel) or
self.frontWheel.isDeepKindOf(AluminumWheel))

context RacingBikeConfiguration
inv O1 3–racingForkFrame: self.fork.isDeepKindOf(RacingFork) and
self.frame.isDeepKindOf(ProfessionalRacingFrame)

context RacingBikeConfiguration
inv O1 4–mudMount: not (self.fork.isDeepKindOf(SupensionFork)) and
self.mudMount -> size() = 0

context ProfessionalRacingBikeConfiguration
inv O1 5–minimumWeight: self.frame.weight >= 5200

context ProfessionalRacingBikeConfiguration
inv O1 6–certification: self.organisation -> size() >= 1

context CityBikeConfiguration
inv O1 7–rearSuspension: self.frame.rearSupension -> size() = 0
```

3.4 Bicycle Configurations Level - O_2

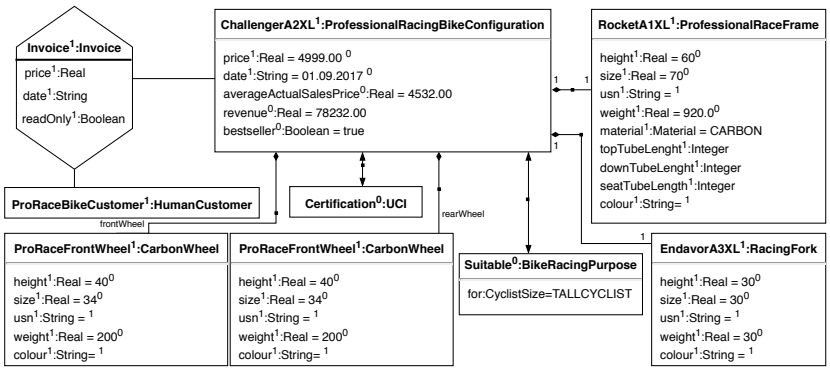

Fig. 3. Bicycle Configuration Model (O_2)

Figure 3 shows the example bicycle configuration described in the Bicycle Challenge. This configuration, called `ChallengerA2XL` is an instance of the `ProfessionalRacingBike` category and

has a regular sales price of 4999.00. The `averageActualSalesPrice` is determined by the derive constraint $O_0 2$ to have a value of 4349.25. The derived value for the `revenue` attribute (for 2017) is 78232.0 as described in the Challenge description. It is also the best selling configuration at this level. The connected components represent the minimum configuration satisfying the constraints defined at the level above. For instance, the connected wheels both have the same size and can be distinguished by the instance of the connection. The front wheel will be connected to the front wheel connection instance and the rear wheel to the rear wheel connection.

3.5 Bicycle Instances Level - O_3

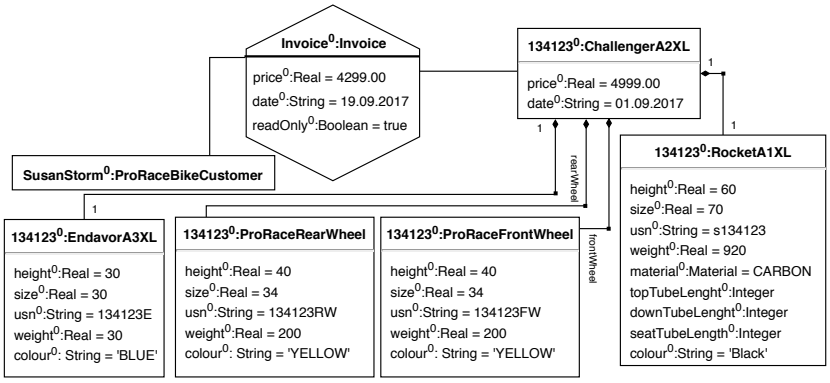

Fig. 4. Bicycle Instance Level (O_3)

Figure 4 shows the bicycle configuration instances described in the Challenge description at the lowest (most concrete) level of the deep model. The `Invoice` connection captures the sales transaction and is the source of information for the derive constraints for the attributes `revenue`, `bestseller` and `averageActualSalesPrice`. The bicycle concerned is an instance of `ChallengerA2XL` sold for a price of 4299.00 instead of the regular price of 4999.00 as shown in the `Invoice` connection. The values of the attribute of the clabjects derived from `Component` and `Product` cannot be changed at this level because their mutability is set to zero at the level above. Except for the attributes `colour` and `usn` of the `Component` clabject.

4 Discussion

Having presented our solution to the Bicycle Challenge, in the section we discuss its pros and cons according to the criteria outlined in the Challenge description.

4.1 Mandatory Comparison Criteria

Basic Modeling Constructs. The basic benefit of multi-level modeling is that it simplifies the modeling of domains inherently consisting of instantiation chains having two or more ontological classification relationships, such as the Bicycle Challenge. There is no claim that multi-level modeling makes it possible to represent scenarios that cannot be modeled using traditional two-level

languages, the basic value proposition is that it does so with less accidental complexity (e.g. by obviating the need for workaround patterns such as the type-object patterns etc.) [13]. The deep model presented in this paper demonstrates this by capturing all information in the Bicycle Challenge in the style of the UML/MOF core infrastructure language, but with minimal linguistic overhead.

Three features of the approach are primarily responsible for achieving this minimality. The first is the compaction of elements that play both type and instance roles into elements represented using a single amalgamated linguistic concept – "clabject". The second is the separation of ontological classification from linguistic classification, and the use of a modeling architecture that allows the former to be concisely represented within an arbitrary number of "levels" spanned by the latter. In our deep model, ontological classification is represented using the tradition UML ":" notion. The third is the use of a single linguistic attribute of clabjects and attributes, called potency, to capture their "instantiatability/changeability" over multiple levels. While the first two features are common to most multi-level modeling approaches, potency is unique to deep modeling. It helps minimize linguistic overhead by capturing important properties of a model without the need for additional linguistic constructs. For example, the three forms of potency in ***MELANEE*** – clabject potency, attribute potency (a.k.a. durability) and value potency (a.k.a. mutability) obviate the need for features such as abstract classes, tags/tag values, static variables, power types and dependencies in the UML. For example, Figure 1 shows the use of potency 0 clabjects to represent abstract classes, durability 2 attributes to represent tags, and mutability 2 attribute values to represent tag values.

Levels and Layers. In the deep modeling approach supported by ***MELANEE***, levels are defined by ontological classification (i.e. instance-of) relationships according to the tenet of strictness. In essence, this states that the ontological types of a clabject must reside at the level above that clabject. However, the "level at which a clabject" resides does not necessarily correspond to its order (i.e. instantiation power) as in most multi-level modeling approaches [8]. This is reflected by the fact that although the direct type of a clabject must have a potency that is one higher than that of the clabject, indirect types (i.e. superytpes of the direct type) can have a potency lower than that of the direct type as illustrated in Figure 1. All inheritance (specialization-of) relationships must reside within a single ontological level in a ***MELANEE*** deep model.

This approach has the advantage that abstractness can be represented in a nuanced and minimalistic way using potency as shown in Figure 1. How it also arguably has the disadvantage that "unnecessary" clabjects have to be included in levels when an abstraction needs to be instantiated at more than one level [9]. For example, since the abstraction `HumanCustomer` is defined at level O_0, so that it can inherit from `Person`, but is "used" at O_3 where actual human customers are represented, the intermediate levels also have to contain offspring of `HumanCustomer`. This problem could be alleviated by cross-level structural relationships of the kind suggested in [8], but this would require a relaxation of the potency rules so that base-types could be more frequently used.

Abstraction. By making ontological classification a first class citizen of models alongside specialization, deep modeling allows model content to be organized into abstraction levels that best match the concerns occurring in the domain of discourse. This is clearly illustrated by our Bicycle Challenge deep model, where the O_0 level focuses on the abstract concerns of product structure and sales, the O_1 focuses on describing bicycles categories as instances of products, O_2 focuses on describing specific bicycle configurations as instances of bicycle categories, and O_3 represents specific instances of bicycle configurations.

Reuse. The information in a deep model can be reused in two ways based on the two fundamental relationships of ontological classification and specialization. Since the top level O_0 metamodel is

independent of the bicycle domain, it can be used to support new product areas through classification simply by instantiating new models from it. Alternatively, the existing models derived from O_0 can be customized by specializing the existing clabjects with more refined versions. To further facilitate reuse, *MELANEE* also supports model importation and composition [2].

Cross-level relationships. *MELANEE's* deep modeling approach strictly forbids any kind of relationship other than ontological classification relationships between levels. This requires modelers to be disciplined and create models which are organized into clear abstraction levels. However, it arguably has the disadvantage that additional clabjects needed to be included in certain levels to ensure that offspring of a certain concept exist at the correct level to avoid level-crossing relationships.

Cross-level constraints. The deep OCL dialect used to define the constraints in our solution is aware of, and can exploit, the two distinct modeling dimensions and multiple levels that occur with the ontological dimension [12]. As illustrated in Section 3, this constraint language can be used to define level-spanning (i.e. "reflective") constraints which adjust the rules by which levels relate to each other, as well as multi-level constraints that govern the contents of the offspring of a clabject over an arbitrary number of levels

Integrity mechanisms. The ability to change any level of a deep model on-the-fly, in real time, offers a great deal of power and flexibility when used correctly. However, it also means that simple changes can have a dramatic impact on the correctness of a deep model. *MELANEE* therefore contains a built-in emendation service [3] which checks the impact of any changes made by users and (a) automatically updates the rest of the model to be consistent with them and/or (b) checks whether they have consequences intended by the users by displaying them and asking for conformation.

Code Generation and Consistency. *MELANEE* does not have any in-built code generation capabilities, but since it is a part of the ECLIPSE environment, its rich set of development plug-ins can be used to map *MELANEE* models to other IT artifacts. The *MELANEE* toolkit is also accompanied by a specially designed transformation language, deep ATL [5], which is aware of and supports transformations between deep models.

4.2 Additional Comparison Criteria

In this paper, we have presented our solution to the Bicycle Challenge using the standard, general purpose concrete syntax for the LML. However, *MELANEE* also allows users to define an arbitrary number of alternative concrete syntaxes for (offspring of) an ontological level which can have multiple formats (i.e. graphical, text-based, form-based tabular). Not only can these different notations be used side-by-side with each other and with the general syntax to provide multiple, concurrent views on a deep model, they be can automatically applied at any depth below the level where they are defined. Furthermore, the rendering of a clabject can also be made context-sensitive so that different symbols are used in different situations. This ability to define deep, context-sensitive, domain-specific language within the ontological levels of a deep model allows a wide range of advanced business services and tools to be supported. For example, because deep models contain instance data as well as type data (i.e. they embody models@run-time), performance data can easily be collected and displayed in the style of a business analytics tool. Alternatively, by defining a business process modeling language at the top level, business processes can not only be represented in a domain specific syntax at the level below, but the execution of the instances of these process can be displayed at run time using the same notation. Ultimately, this capability allows *MELANEE* to support the full-scale simulation of deep models and the creation of model-driven tools [4].

5 Conclusion

This paper has presented a full solution to the MULTI 2018 Bicycle Challenge taking the form of a deep model in *MELANEE*. By capturing all mandatory and optional requirements of the Challenge in a highly concise, precise and easy-to-understand way, the model reinforces the core value proposition of multi-level modeling which is to reduce accidental complexity without loss of clarity. In particular, the model demonstrates that it is possible to clearly and precisely describe complex, multi-level domain scenarios in a UML-like way without resorting to the plethora of auxiliary modeling features utilized by the UML (i.e. stereotypes, tag/tagged values, dependencies, static variables, power types etc.). By treating all levels uniformly using the same core relational concepts (inheritance, containments, classification etc.) the approach also provides a new dimension of flexibility to modelers in which they can change abstract (i.e. meta) levels as easily as they can change concrete (i.e. instance) levels. It therefore lays the foundation for models to support all phases of a software system's lifecycle (i.e. development, deployment and operations) and bring the vision of models@runtime closer to reality. The next step is to develop a version of the Bicycle Challenge based on traditional two-level modeling technologies to perform a concrete comparison.

References

1. 5th international workshop on multi-level modelling, https://www.wi-inf.uni-duisburg-essen.de/MULTI2018/
2. Atkinson, C., Gerbig, R., Fritzsche, M.: A multi-level approach to modeling language extension in the enterprise systems domain. Information Systems **54**, 289–307 (2015)
3. Atkinson, C., Gerbig, R., Kennel, B.: On-the-fly emendation of multi-level models. In: Vallecillo, A., Tolvanen, J.P., Kindler, E., Störrle, H., Kolovos, D. (eds.) Modelling Foundations and Applications. Lecture Notes in Computer Science, vol. 7349, pp. 194–209. Springer Berlin Heidelberg (2012)
4. Atkinson, C., Gerbig, R., Metzger, N.: On the execution of deep models. In: Mayerhofer, T., Langer, P., Seidewitz, E., Gray, J. (eds.) Proceedings of the 1st International Workshop on Executable Modeling. EXE 2015, vol. 1560, pp. 28–33. CEUR Workshop Proceedings (2015)
5. Atkinson, C., Gerbig, R., Tunjic, C.: Towards multi-level aware model transformations. In: Hu, Z., de Lara, J. (eds.) Theory and Practice of Model Transformations. Lecture Notes in Computer Science, vol. 7307, pp. 208–223. Springer Berlin Heidelberg (2012)
6. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: International Conference on the Unified Modeling Language. pp. 19–33. Springer (2001)
7. Atkinson, C., Kühne, T.: Rearchitecting the uml infrastructure. ACM Trans. Model. Comput. Simul. **12**(4), 290–321 (Oct 2002)
8. de Carvalho, V.A., Almeida, J.P.A.: Toward a well-founded theory for multi-level conceptual modeling. Software and System Modeling **17**(1), 205–231 (2018)
9. De Lara, J., Guerra, E., Cobos, R., Moreno-Llorena, J.: Extending deep meta-modelling for practical model-driven engineering. The Computer Journal **57**(1), 36–58 (2012)
10. Draheim, D.: Reflective constraint writing. In: Transactions on Large-Scale Data-and Knowledge-Centered Systems XXIV, pp. 1–60. Springer (2016)
11. Gerbig, R.: Deep, seamless, multi-format, multi-notation definition and use of domain-specific languages. Verlag Dr. Hut (2017)
12. Lange, A.: dACL: the deep constraint and action language for static and dynamic semantic definition in Melanee. Master's thesis, University of Mannheim (2016), http://ub-madoc.bib.uni-mannheim.de/43490/
13. Lara, J.D., Guerra, E., Cuadrado, J.S.: When and how to use multilevel modelling. ACM Transactions on Software Engineering and Methodology (TOSEM) **24**(2), 12 (2014)
14. Software Engineering Group University Mannheim: Melanee – the deep-modeling domain-specific language workbench (2018), http://www.melanee.org/