An Example Application of a Multi-Level Concrete Syntax Specification with Copy-and-Complete Semantics

Jens Gulden

University of Duisburg-Essen
Universitätsstr. 9, 45141 Essen, Germany
`jens.gulden@uni-due.de`

Abstract. This paper describes an example application of the Topology Type Language (TTL) to define visualizations for conceptual models with multiple type levels. Based on a multi-level example model about the domain of bicycle products, a formal specification for a diagram visualization is developed which visually displays characteristics of bicycles and their components according to the domain model.
The result is an example specification of a diagram visualization that incorporates characteristics of model elements from different type-levels of a domain-specific conceptual multi-level model into one consistent visualization specification that can be reused to visualize entities on different abstraction levels in the domain model.

Keywords: Multi-Level Modeling, Domain-Specific Modeling Language, Diagram, Topology, Visualization, Concrete Syntax

1 A Visual Formalism for Specifying Visualizations

1.1 Overview

The Topology Type Language (TTL) is being developed to serve the purpose of specifying visualizations of diverse kinds for models. It is intended to describe model representations of various visual forms, such as diagrams or graphical user interfaces, together with corresponding human interactions. Especially, the TTL is being developed as an advanced specification formalism for concrete syntaxes of domain-specific modeling languages. Among others, it fulfills three central requirements:

Req. 1: The description of a visualization happens entirely independent from a model, i. e., the visualization can be defined without restrictions imposed by the meta-model of the visualized language.

Req. 2: The specification mechanism can refer to meta-models with multiple type levels. It describes a notion of instantiating topology types in multiple refinement steps which can be mapped to the conceptual type-level hierarchy of a multi-level meta-model.

Req. 3: The approach provides a visual formalism to describe topologies. Although the description of topologies is an abstraction over a visualization and does not describe the visualization itself, the use of a visual language to specify topologies appears appropriate as it promises a higher level of cognitive efficiency and advanced ease-of-use than a textual formalism.

Figure 1 shows the example TTL model which is introduced in this use case demonstration.

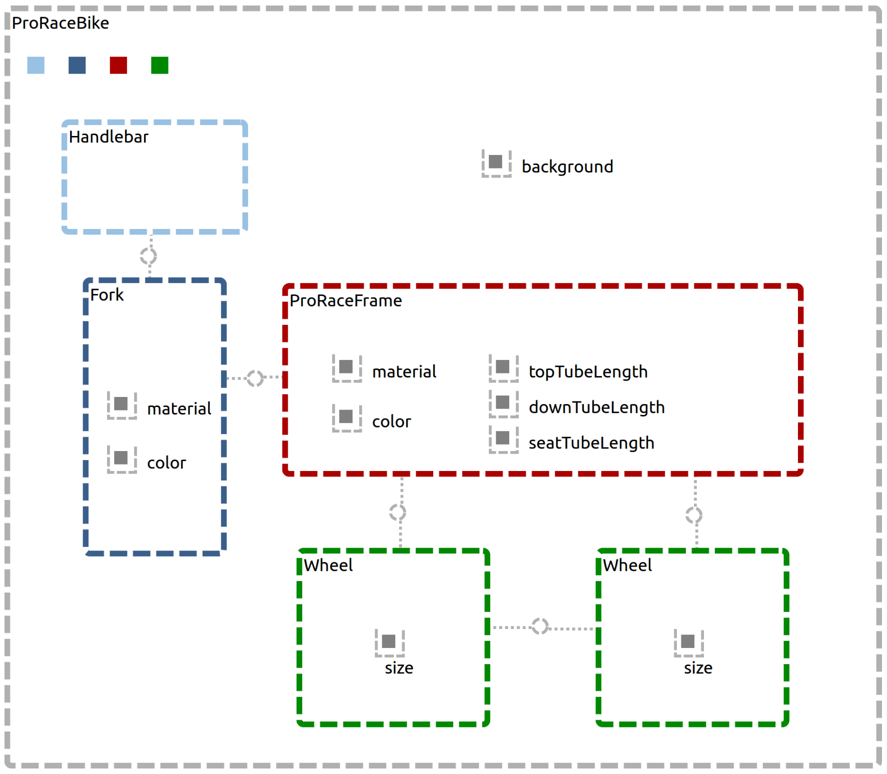

Fig. 1: Example visualization specification for a bike product type

Some central considerations that have guided the development of the TTL are briefly described in the following.

Decoupling Visualization Description from Conceptual Description In fact, it is a major shortcoming of existing approaches such as the Eclipse Graphical Modeling Framework (GMF) [5], that the use of different types of visual elements that determine the visual structure and composition of the syntax, such as line connectors, or nesting elements inside each other, can only be used in combination with fixed meta-model concepts. For example, a visual nesting of things, in which elements are placed inside an outer element, requires the meta-model to contain a composition relationship between the nested concepts. The use of line-connectors to show relationships between elements, on the other hand, demands for non-containment associations to be present between the classes of the connected elements in the meta-model, and cannot be applied to associations that are considered to be compositions from a conceptual point of view – although the decisions whether to model a relationship as association or as composition may be contingent in the modeled domain, and should not influence the shape of possible visualizations that can be specified as a concrete syntax. (Technically, from the point of view of a programming language, associations and composition relationships are implemented the same way anyway.)

As a consequence, it turns out to be necessary for a specification mechanism for model visualizations, to introduce a layer with intermediary specification concepts, which serves to decouple the structure of the described visualizations from the structure of the meta-model that is underlying to the (domain-specific) conceptual language the concepts of which are visualized. The approach provided with the TTL does so by abstracting visualizations to topologies, which can initially be described without any references to a modeling language and its instances, and only at a later stage during the specification of the visualization are bound to concrete model instances (see the following section).

The actual binding between topology content and model content happens by describing *conceptual relations* of model content, and optionally transform model content with the help of *lenses*. These two concepts are not of central relevance for the example shown here, they will thus be described elsewhere.

Instantiation by Copy-and-Complete Semantics The specification language supports a notion of Abstract Areas versus Concrete Areas, and in addition it allows to specify a sequence of Completion Steps that constrain how an Abstract Area is supposed to be transformed to a Concrete Area, in order to make it visually renderable. Among other linkages to multi-layer or multi-step application contexts, these constructs in combination allow to reflect a notion of topology types which over multiple levels of abstractions get instantiated to a concrete topology. As this procedure is applied to a visual formalism, it makes sense to rather describe the differences between types and instances in terms of missing versus provided elements, which is why an operational *copy-and-complete* semantics to turn a type description into an instance, seems appropriate to describe a visual type system.

Following such a *copy-and-complete* procedure, the transformation from abstract to concrete topology type descriptions happens in an n-step transformation process, in which underspecified parts of the topology type description are completed step by step, until no more underspecified parts exist. Figures 7 and 8–1 show the example *copy-and-complete* procedure that is applied to create the topology type description for the example domain model.

1.2 Language Elements

Those language elements of the TTL that are used in the example presented in this paper are briefly explained in the following.

Area The Area concept is the central model element to specify topologies. It provides the fundamental structure element to describe visualizations. Areas are assumed to have a spatial extension and can be nested into each other. Every area has a location that may be relative to other areas, and to its parent area (if exists). Areas are notated by slightly rounded squares with a dashed line border (see Fig.2).

Areas provide templates for renderers. Areas do not define their visual appearances by themselves, but they provide space for renderers to work in. They also provide data for renderers to work on. The notion of a renderer refers to program code that uses the topology type description and bound values from the domain model as input to actually display visualizations on top of a concrete graphics rendering technology. From the conceptual point of view of the TTL, renderers can be implemented using arbitrary technology. Different renderers may be responsible for displaying the same topology type description via different document formats, or on different devices. The wide range of realization options for renderers is not in the focus of this paper and will be discussed elsewhere.

An area can be associated with a type. Visually, types are distinguished by different colors. The definition of available types is indicated by small squares in the top section of an area.

Figure 2 shows Area notation elements nested inside each other, with the parent area defining two distinct types, that are applied to the nested areas.

Further details on the specification of Areas are out of the scope of this paper and will be discussed elsewhere.

Locator A Locator serves the purpose to formally describe spatial relationships between Areas. The visual presence of a Locator in a model, that connects two or more areas with each other, semantically only means that there is an explicit statement on how the respective areas are related to each other. The actual kind of spatial relationship is non-visually specified via the properties-sheet of the Locator. There are multiple options to actually implement a formal description scheme for spatial relations. One possibility is to recur to a 2D spatial version Allen's interval algebra, as it is elaborated in [16], which list all possible kinds of spatial relations that bodies can have on a diagram plane, e. g., overlapping, touching containing, etc.

The spatial placement of a Locator inside an area and in relation to its connected areas, as it appears in a TTL diagram, is a pure visual hint for the topology modeler that may of may not visually resemble the actual placement the Locator is expressing.

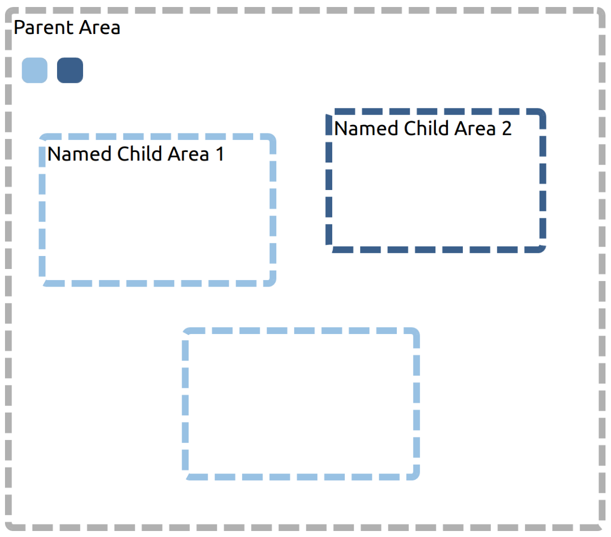

Fig. 2: Notation of Area elements

For the purposes of the example presented here, a simple notion of locators that specify an absolute distance and direction between the two connected areas is sufficient.

The visual notation of a Locator is shown in Fig. 3.

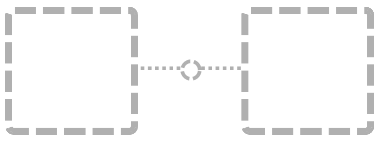

Fig. 3: Notation of a Locator element (middle circle) with Locator Links to two areas (dashed lines)

Abstract Areas versus Concrete Areas In order to provide a notion of instantiability of abstract topology types to concrete ones, the notion of regarding an incomplete specification of a topology as abstract, and a fully specified topology as concrete which can serve as input to a renderer, can be broken down to individual Areas. An Area which is not yet fully specified to serve as input to a renderer is considered abstract, while an Area which provides all necessary information to be rendered is concrete.

There are three ways to make sure an Area can be rendered, i. e., to *concretize* an Abstract Area to a Concrete Area:

– Children Areas are added (every Area that contains at least one child Area is considered to be concrete, because a default renderer can be applied)
– Conceptual Relation Specifications are added to all Conceptual Relation placeholders that an Area contains (Conceptual Relation Specifications bind elements from the data population of an area to parameters of renderers)
– An Area Reference to a Concrete Area is added, which has the effect that the referenced Area is filled-in in the place of the referencing one

The first two options can be combined, given that the configured renderer both processes children Areas and Conceptual Relations. In the special case that a Renderer which has no parameters at all is assigned (e. g., a visual decorator), an Area is also considered to be concrete.

The notation of Conceptual Relations and Conceptual Relation Specifications is shown in Fig. 4. Further details about the Conceptual Relation concept are not required for the example case presented here and will be discussed elsewhere.

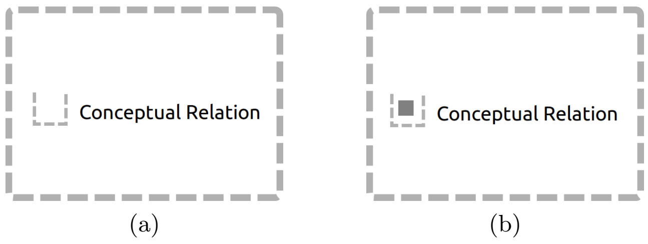

Fig. 4: Notation of the Conceptual Relation element inside an Area element, (a) abstract requirement, (b) filled in with concrete Conceptual Relation Specification

Completion Step Abstract Areas can be assigned with Completion Step tags, that allow to specify a minimum and / or maximum number of concretization steps, until the Abstract Area either has be made concrete, or optionally is removed. In general, Completion Step tags allow to specify any finite sequence of modes that subsequent concretization steps have to conform to.

The three distinguishable modes of how to perform a single completion step on an Abstract Area are:

– **Preserve**: the Abstract Area is not completed and remains abstract in the current completion step
– **Optional**: the Abstract Area may be completed, remain, or be deleted in the current completion step
– **Complete**: the Abstract Area must be completed with a concrete topology description, either by assigning a renderer to the area, describe a subtopology in the area, or reference another area which is concrete

Figure 5 shows the notation of Completion Step tags as a sequence of differently drawn small circles in the upper right top part of an area symbol, together with a brief legend explaining the three different modes of appearances of the circles.

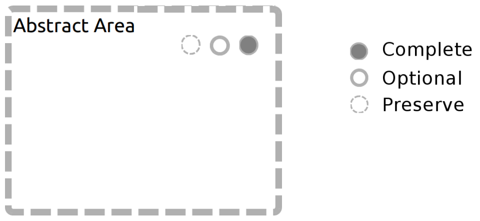

Fig. 5: Notation of Completion Step tags (shown as circles in the upper-right corner of an area)

The sequence of Completion Step tags, from left to right, represents the upcoming concretization steps that the topology type description will undergo. The very next step is configured by the left-most Completion Step tag in the sequence. When a topology type description is transformed to a next concretion step (a procedure that can be seen in analogy to instantiating a type entity into an instance on one abstraction level lower), all Areas get copied, and the left-most Completion Step tag is removed from the sequence in any of the areas.

The ability to impose constraints on anticipated future concretization steps provides a mechanism which stands in analogy to the specification of intrinsic features in a multi-level modeling language [7]. This means, the sequence of concretization steps can be defined along the instantiation levels of entities from a multi-level conceptual domain model. Using Completion Step tags, the demand for when to fill in Conceptual Relation specifications can be delayed in a controlled way. A topology type definition can make use of this to define Conceptual Relations on the basis of intrinsic attributes on higher levels of conceptual abstractions in the domain model, for which the concrete values will later be derived from slot values of entities on lower levels. This makes the TTL a visualization specification approach that integrates the description of visualizations for conceptual entities on different levels of conceptual abstractions into one unified specification mechanism. It allows for an efficient declaration of concrete syntaxes for domain-specific type models, together with concrete syntaxes for their instance models across multiple type levels.

2 Example Application of a Bike Product Visualization Based on a Multi-Level Conceptual Domain Model

The example domain model that conceptually describes the bicycle product domain is displayed in Fig. 6. It has been created with the multi-level modeling language FMML^x [7] as a contribution to the MULTI 2017 Challenge [3]. The design decisions that led to the conceptual domain model are described in [9].

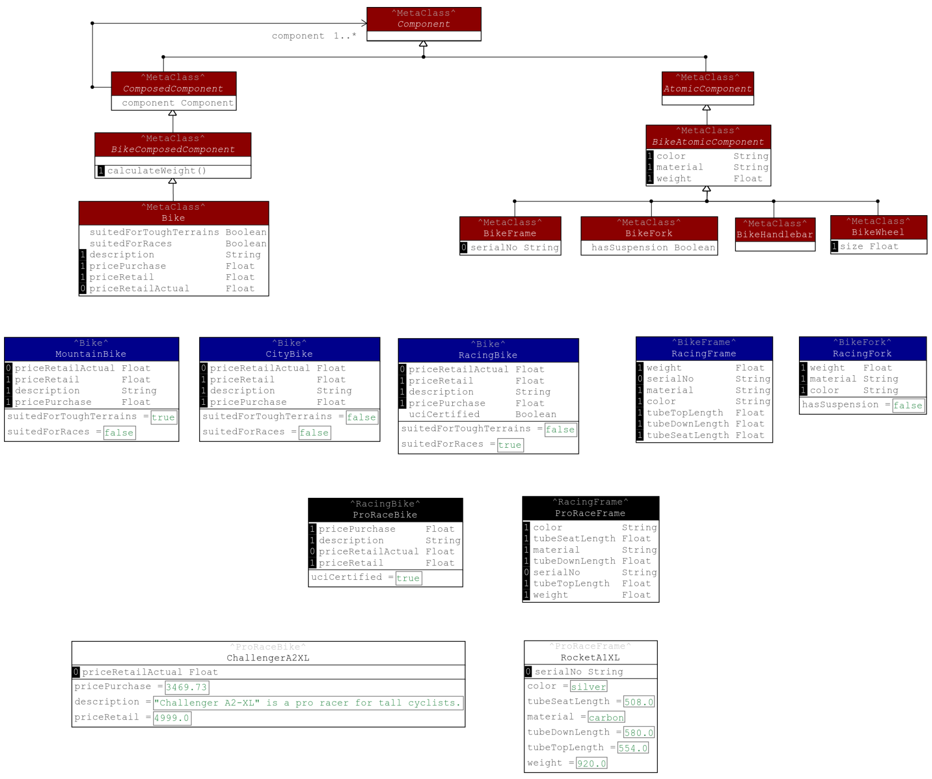

Fig. 6: Example Conceptual Model of a Bike Product Domain

The model in the TTL shown in Fig. 1 is the result of a multi-step transformation from a more general TTL model to a concrete visualization description for “Pro Racing Bikes”. The procedure of the transformation is described in the following, starting with the initial TTL model shown in Fig. 7 that generally describes the notion of any bike visualization without a connection to concrete data to be rendered.

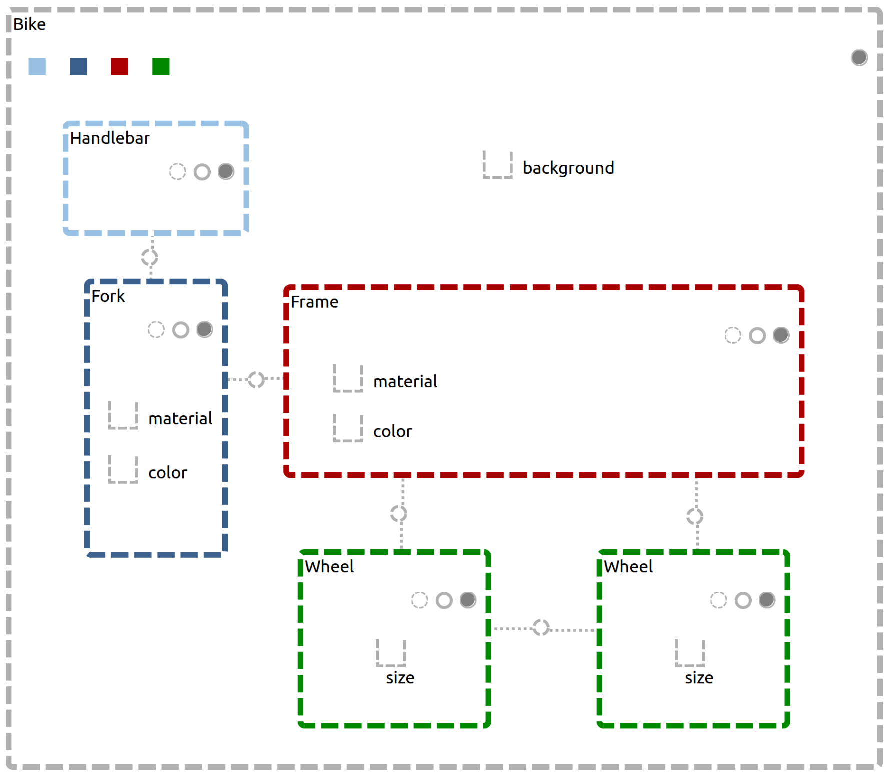

Fig. 7: Example visualization specification for the generic notion of a bike, as modeled on the highest abstraction level in the domain model

2.1 Initial model

The TTL model in Fig. 7 provides a general description of what a visualization of a bike product could be composed of. The description is independent from any binding to model content yet, but by specifying several empty Conceptual Relationship elements in its area elements, it already suggests what domain characteristics should influence a resulting rendered visualization.

The Completion Step elements in the upper-right corners of each area part suggest how to further complete this topology type description in the next concretization step. The outer "Bike" area specifies a single further Completion Step, which demands to concretize this area specification in the next subsequent step. The other areas each define three subsequent concretization steps, with the first demanding to leave the area untouched in the next step.

2.2 First concretization step

As a consequence, in the next step the outer "Bike" area is made concrete by filling in the previously empty Conceptual Relation element "background" with a binding that provides input data from the visualized model as input for the renderer that is configured for the "Bike" area. In case of the example at hand, this could mean that either based on the class name of a bike entity to be displayed, or as a result of combining the suitedForToughTerrains and suitedForRaces slot values, a suitable name for a background image is calculated. E. g., a renderer could be configured which displays a background image of a mountain scenery for mountain bikes, a street scenery for city bikes, and a racing track for racing bikes. There are diverse options for realizing such a binding on the implementation level, further details are not discussed here.

Figure 8 shows the topology model after the first concretion step has been performed.

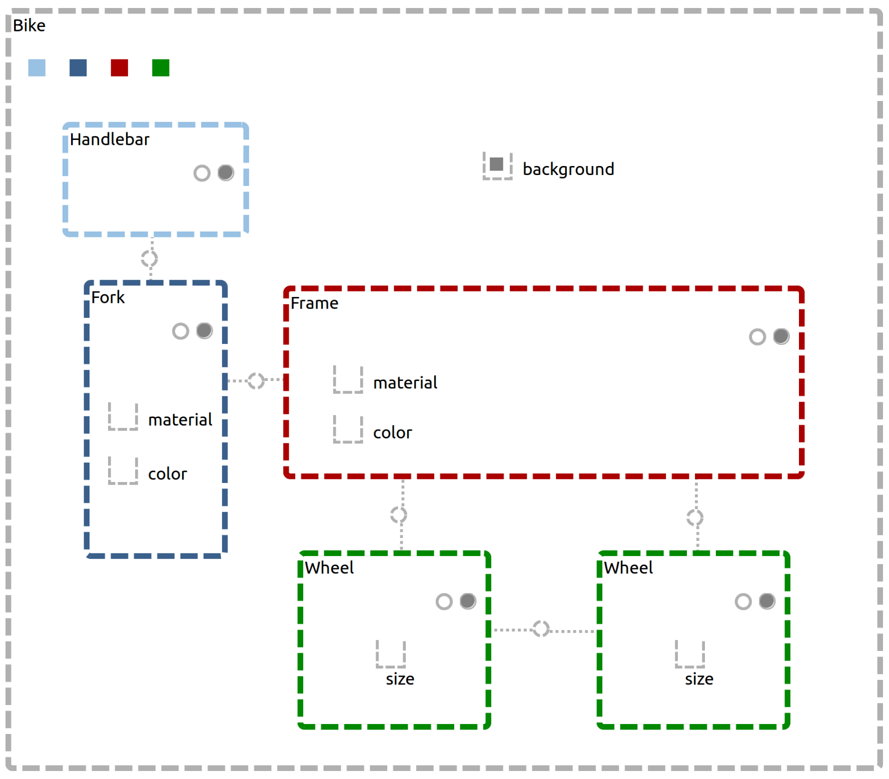

Fig. 8: Example visualization specification distinguishing the notion of general types of bikes (racing/mountain/city), as modeled on the second-highest abstraction level in the domain model

If at this point also renderers are attached to the other areas, which could render a generic visualization of the respective elements they are to display even without concrete input values (e. g., by using pre-defined default values as long as the Conceptual Relations for the respective area are not fully specified), then the visualization description could already be used for rendering a generic, product-independent, visualization of a bike.

2.3 Second concretization step

To provide a further refinement of the visualization specification, the next concretization step reuses the topology type description in Fig. 8 and enhances it in order to visualize specific characteristics of racing bikes described in the domain model. To do so, additional Conceptual Relations are added to the "Frame" area which represent the domain fact that the frames of racing bikes are described by three lengths values. Adding further Conceptual Relations is possible, because the Conretization Step specifications for the current step allows optional modification to the area. The resulting topology type description after this step is displayed in Fig. 9.

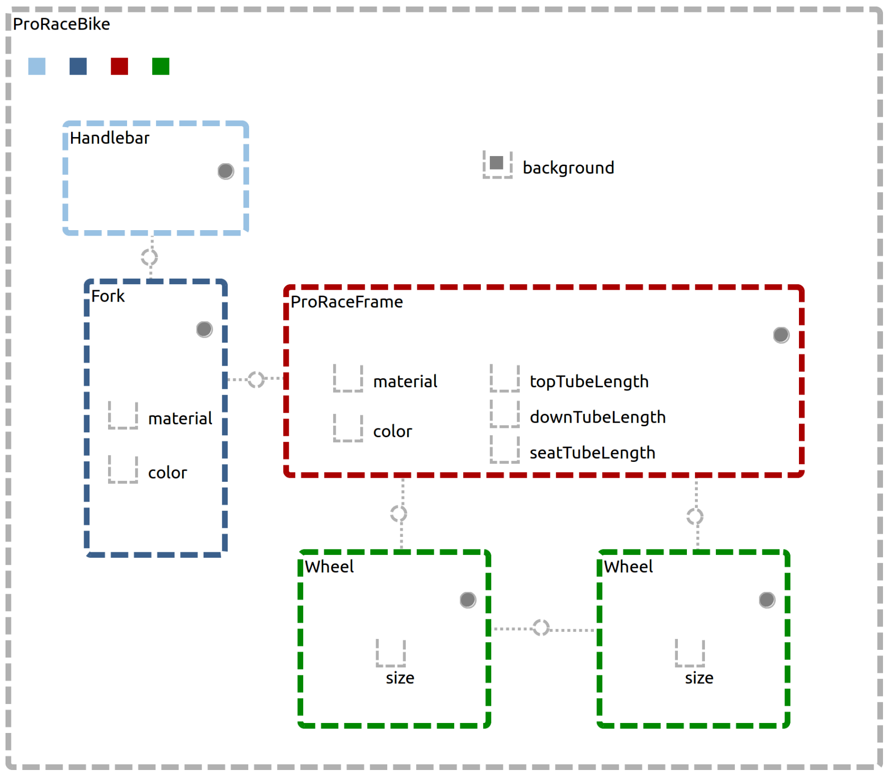

Fig. 9: Example visualization specification incorporating characteristics of "Pro Racing Bikes", as modeled on level 2 in the domain model

2.4 Third concretization step

The remaining Concretization Step elements in the model in Fig. 9 all demand for a completion of the yet underspecified areas. As a consequence, the final modifications to the model in the third concretization step consist of filling in the empty Conceptual Relation placeholders with bindings to concrete input value that can be derived from domain model entities on abstraction level 1. According to the declarations of intrinsic attributes in the domain model, this is the level where the information is present that distinguishes visual characteristics of different bike products.

Figure 1 shows the resulting concrete renderable topology type description for "Pro Racing Bikes", as described in the domain model.

3 Related Work

Specifying concrete syntaxes for visual diagram languages is pivotal to the design of domain-specific languages (DSLs). As a consequence, most of the DSL creation approaches and tools available offer means for defining concrete diagram syntaxes. Three well known representatives of the category of meta-modeling environments which offer support in visual language creation are MetaEdit+ [12], the Eclipse Graphical Modeling Framework (GMF) [5], and Sirius [4]. These tooling environments offer mechanisms to specify conceptual features of a domain, e. g., with a meta-model, together with functionality to define visual representations for the domain concepts. MetaEdit+ includes a simple graphical icon editor that allows to paint graphical symbols to represent domain concepts. GMF also supports the definition of graphical symbols composed out of drawing primitives, however, the specification of the symbols happens non-visually in a tree-view editor. Sirius also uses a tree-view configuration editor for all parameters of its diagram definitions, with the significant difference to GMF that the concrete syntax definition is interpreted at run-time without the need for code generation. Any changes that are made to a Sirius diagram definition become immediately visible in a corresponding editor.

The general approach in these tools is to enhance the conceptual description in the meta-model with additional information about the visualization that gets attached to the meta-model elements. This happens either by directly attaching information about how to visualize a concept in the meta-model (e. g., by the use of annotations), or by employing a mapping model that externally attaches additional information to meta-model elements. In both cases, one has to be aware that a direct annotation of meta-model elements limits the range of possible visualizations to describe, as the meta-model structure pre-forms the possible structure of visualizations [10, 11]. None of the approaches embedded in existing tools has been developed beginning with theoretical considerations about the demands towards an optimal visualization specification mechanism. The TTL aims to overcome these limitations.

Despite the prominent role of diagram languages in Information Systems, theoretic research about concrete syntax specification is just about to be established as a relevant perspective on the core objectives of the discipline. Considerations about the "Physics of Notation" [15] provide one source in Information Systems science that summarizes fundamental principles of designing visual diagram languages. Diverse points of critique have been brought forward against the narrow perspective taken in by [15]. Especially the potential for leveraging the capabilities of the human visual perception apparatus, which allows for fast, parallel, and scalable cognitive processing of visual pattern structures, is not sufficiently taken into account [20, 21].

Beyond the Information Systems discipline, a wider range of scientific contributions about information visualization can be found. Classical work about diagrams from before the age of computers has been contributed by [1, 18, 19]. More recent work about the effectiveness of interactive visualizations originates from fields such as interaction design and journalism [2, 13, 14, 17], or information dashboard design [6]. The research demand for incorporating the perspective on cognitive efficiency into Information Systems research has been pointed out as well [10, 11].

4 Conclusion

The example developed in this paper has given an impression of how a visual formalism for specifying visualization types that can represent model content on multiple levels of abstraction can work. Core elements of the visual Topology Type Language (TTL) have been introduced, without, however, going too much into details to keep the example description appropriately compact.

The TTL integrates the description of visualizations for conceptual entities on different levels of conceptual abstractions into one unified specification. This allows for an efficient reuse of existing concrete syntaxes, and makes it possible to specify visual languages for domain-specific type models as well as their instance models across multiple levels in the same place.

Other aspects of the TTL with respect to its applicability to presentation and interaction schemes beyond mere diagram visualizations, toward describing entire applications' user interface presentation and interactions, will be part of future work. The TTL might as well be suited for use in self-referential enterprise system scenarios [8], where dynamically configurable views on instance models serve both as tools for control and analysis. The TTL could play a role in such a setting by providing a visual specification mechanism which allows to define instance visualizations at run-time based on existing reusable specifications of type-level visualizations.

References

1. J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps.* Walter de Gruyter, Berlin, 1974.
2. Albero Cairo. *The Functional Art.* Voices That Matter. Pearson Education, New York, 2010.

3. Tony Clark, Ulrich Frank, and Manuel Wimmer. The bicycle challenge of the multi 2017 workshop on the models 2017 conference, austin, texas, sept 17-22, 2017.
4. Eclipse Foundation. Eclipse sirius. https://eclipse.org/sirius/.
5. Eclipse Foundation. Graphical modeling framework (gmf). http://www.eclipse.org/modeling/gmf/.
6. Stephen Few. *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly, Sebastopol, CA, 2006.
7. Ulrich Frank. Multi-level modeling - toward a new paradigm of conceptual modeling and information systems design. *Business & Information Systems Engineering (BISE)*, 6(3), 2014.
8. Ulrich Frank and Stefan Strecker. Beyond erp systems: An outline of self-referential enterprise systems. Technical Report 31, ICB Institute for Computer Science and Business Information Systems, University of Duisburg-Essen, Essen, April 2009.
9. Jens Gulden. Multi-level domain-specific modeling with the fmmlx – a contribution to the multi 2017 challenge. 2017. *Under Review.*
10. Jens Gulden and Hajo A. Reijers. Toward advanced visualization techniques for conceptual modeling. In Janis Grabis and Kurt Sandkuhl, editors, *Proceedings of the CAiSE Forum 2015 Stockholm, Sweden, June 8-12, 2015*, CEUR Workshop Proceedings. CEUR, 2015.
11. Jens Gulden, Dirk van der Linden, and Banu Aysolmaz. Requirements for research on visualizations in information systems engineering. In *Proceedings of the ENASE Conference 2016, April 27-28 2016, Rome*, 2016.
12. Steven Kelly and Juha-Pekka Tolvanen. *Domain Specific Modeling: enabling full code-generation.* Wiley, 2008.
13. Andy Kirk. *Data Visualization: a successful design process.* Packt Publishing, Birmingham, 2012.
14. Isabel Meirelles. *Design for Information.* Rockport Publishers, Beverly (MA), 2013.
15. Daniel L. Moody. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 11/12 2009.
16. Mohammad Nabil, John Shepherd, and Anne H. H. Ngu. *2D projection interval relationships: A symbolic representation of spatial relationships*, pages 292–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.
17. Robert Spence. *Information Visualization (2nd edition).* Prentice Hall, Upper Saddle River, 2007.
18. E. R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, Connecticut, 1983.
19. E. R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Connecticut, 1990.
20. Dirk Van Der Linden and Irit Hadar. Cognitive effectiveness of conceptual modeling languages: Examining professional modelers. In *Empirical Requirements Engineering (EmpiRE), 2015 IEEE Fifth International Workshop on*, pages 9–12. IEEE, 2015.
21. Dirk van der Linden, Anna Zamansky, and Irit Hadar. A framework for improving the verifiability of visual notation design grounded in the physics of notations. In *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE 2017), Lisboa, Portugal*, 2017.