Towards Model Transformation with Structural Level-spanning Patterns

Sándor Bácsi[a] and Gergely Mezei[b]
Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Muegyetem rkp. 3., Budapest, Hungary

Keywords: Multi-level Modeling, Multi-layer Modeling, Multi-level Transformation, Model-driven Engineering, MDE, Deep Instantiation, DMLA, Level-blind.

Abstract: In MDE, models are meant to be transformed, thus model transformation is one of the basic pillars of MDE. Besides generating source code, transformation is often used to refactor or optimize the models. While the theory and application of graph transformations are active research fields for many years now, existing approaches focus mainly on classic two-level meta-modeling setups. It is a promising research direction however to use graph pattern-based transformations on multi-level models as well. This position paper proposes a textual model transformation language — the DMLA Transformation Language (DTL) — which enables the definition of level-spanning transformation rules in a level-blind multi-layer environment.

1 INTRODUCTION

Model transformations are one of the key pillars of model-driven engineering (MDE) (Sendall and Kozaczynski, 2003) and are key to the productivity and flexibility that make model-driven development so attractive. This is reflected in the rapidly growing interest in model transformations in academia and the increasing sophistication of the transformation capabilities offered by leading modeling environments, such as the Eclipse Modeling Framework (EMF) (Steinberg et al., 2009). While the theory and application of model transformations are active research fields for decades, the most contemporary model transformation technologies suffer from the same fundamental weakness as the modeling languages are based on the restriction to a two-level modeling architecture that accommodates only one pair of classification levels, the meta-model (types) and model (instances). This makes it difficult for models, and transformation languages building on them, to handle scenarios when there are more than just two classification levels in a domain of interest without introducing the phenomenon, called accidental complexity (Atkinson and Kühne, 2008) into models and thus transformations.

Over the last few years, a paradigm for model organization has emerged that supports an arbitrary number of modeling levels, called multi-level modeling (Atkinson and Kühne, 2001). Multi-level modeling aims to solve the shortcomings of classic (meta-)modeling approaches such as OMG's four-level Meta-Object Facility (MOF) (MOF, 2005).

Multi-level modeling allows for an unlimited number of instantiation levels as opposed to the four levels prescribed by OMG. One of the main goals of multi-level modeling is to reduce accidental complexity, which refers to parts of the solution needed only to express its multi-level nature, instead of describing the domain in question (Atkinson and Kühne, 2008). A good example of this is the application of the Item Description (Coad, 1992) pattern to describe multiple domain levels in object-oriented languages. The main idea behind the pattern is that objects play the role of classes, and as such, type-related information can be encoded in them. This means that classification is replaced by association, which — in addition to objects representing class information — leads to accidental complexity. Compared to classic modeling approaches, multi-level models are often reduced in size, are more compact, and describe the target domain more accurately. Therefore, many domains can benefit from multi-level modeling. In recent years, several research groups have developed prototype realizations and applications of multi-level modeling environments, for example Melanee (Atkinson and Gerbig, 2016), MetaDepth (de Lara and Guerra, 2010), MultEcore (Macías et al., 2018) or XModeler (Frank, 2014).

[a] https://orcid.org/0000-0002-4814-6979
[b] https://orcid.org/0000-0001-9464-7128

Bácsi, S. and Mezei, G.
Towards Model Transformation with Structural Level-spanning Patterns.
DOI: 10.5220/0010918800003119
In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 352-357
ISBN: 978-989-758-550-0; ISSN: 2184-4348

There exist many efficient methodologies to find and replace patterns in classic meta-models, but these methodologies suppose that all elements of the pattern are on the same abstraction level. It is a promising direction to use graph pattern-based transformations on multi-level models as well. Our research aims at this goal: creating an approach that supports model transformation consisting of level-spanning patterns. These patterns could capture domain concepts across multiple abstraction levels, where the elements can refer to each other independently of their abstraction level. A typical modeling scenario for this is when we have a domain concept containing several components, some of which are concrete, while others are more abstract, i.e. not yet specified completely.

In this position paper, we propose a model transformation language — DMLA Transformation Language (*DTL*) — which can capture level-spanning domain concepts in a multi-layer context and supports model-to-model (M2M) transformations in our multi-layer modeling framework, the Dynamic Multi-Layer Algebra (DMLA) (Urbán et al., 2018; Urbán et al., 2017). The feasibility of the language is demonstrated by the manipulation of concrete domain models.

The structure of the paper is as follows. Section 2 is dedicated to discuss the related work showing how other multi-level approaches support model transformation, while Section 3 describes the basics of our multi-layer framework, DMLA. In Section 4 we elaborate on our motivation and compare multi-layer and classic two-level model transformations in general. Section 5 contains our contribution, where we present *DTL* using illustrative examples. Concluding remarks are outlined in Section 6.

2 RELATED WORK

The theory and practise of graph transformations are well-studied, and the concept of model transformations applied to multi-level modeling is not novel. Earlier works in the area have extended existing model transformation languages to be able to manipulate multi-level models and model hierarchies.

An important concern when discussing multi-level approaches is the basis upon how they handle levels (Atkinson et al., 2014). Approaches that acknowledge the existence of explicit modeling levels are often referred to as *level-adjuvant*. Similar yet highly different approaches have also emerged in the form of *level-blind* approaches, which do not acknowledge explicit instantiation levels, although they can still implicitly implement the concept of levels. To date, transformations between multi-level models represented using *level-blind* frameworks have barely been explored. Previous works have exclusively focused on *level-adjuvant* model transformation methodologies that enforce "strict metamodeling" principles (Atkinson, 1997). In (Atkinson et al., 2013), the authors adapt ATL (Jouault et al., 2008) to manipulate multi-level models built with the level-adjuvant Melanee tool (Atkinson and Gerbig, 2016). In a similar manner, (Lara et al., 2013) proposes the adaptation of ETL (Kolovos et al., 2008) and other languages from the Epsilon family for the application of model transformation rules into multi-level hierarchies created with MetaDepth (de Lara and Guerra, 2010). A much more formal approach can be found in (Wolter et al., 2020). The authors represent multi-level models by multi-level typed graphs whose manipulation and transformation are carried out by multi-level typed graph transformation rules.

As opposed to the aforementioned level-adjuvant approaches, our multi-layer approach, DMLA can be considered level-blind since levels are not explicitly modeled. To our knowledge, no prior studies have examined applying model transformation in a level-blind environment. This position paper proposes a textual model transformation language — the DMLA Transformation Language (*DTL*) — which enables the simple definition of transformation rules in a level-blind environment.

3 DMLA

The main goal of Dynamic Multi-Layer Algebra (DMLA) (Urbán et al., 2018; Urbán et al., 2017) is to offer an environment in which one can model concepts, connect these concepts and then refine them step-by-step following a top-down method. DMLA aims to provide a high-level of flexibility, but at the same time a rigorous validation mechanism during refinement. Typically, one initially only has a vague conceptualization of the domain concepts and only gradually obtains a more concrete understanding of them. Modeling in DMLA aims to follow and aid this process by providing a multi-layer modeling environment.

In DMLA, the main relationship between elements at different levels of abstraction is "refinement" which is used to gradually constrain concepts. Please note that refinement in DMLA is special in nature, its semantics is completely different from the classical object-oriented "specialization" or "instantiation" relations. Refinement relates a DMLA entity to its "meta" and the framework automatically validates if there is indeed a valid refinement relationship be-

tween the two entities. Note that while each entity has a meta-entity, DMLA is a "level-blind" approach (Atkinson et al., 2014) since levels are not explicitly modeled. DMLA does not use a global level organisation that requires refinement relationships to align with each other in any way. Each modeled entity can refer to any other entity along the meta-hierarchy, as long as the validation rules are not violated.

Modeling entities, which have an internal structure, describe their setup by *slots*. The slots of an entity set up its structure, for example a *Bicycle* has a *Wheels* slot containing a list of references to *Wheel*s. At the topmost abstraction level, one does not usually have much information about the exact structure and the details of an entity, therefore slots are used merely as placeholders. Further down one can refine the entity: add new slots, refine slots or omit them. Only optional slots (slots with cardinality 0..n) can be omitted from an entity. Each of these decisions are validated against the validation formulae and constraints describing the refinement rules of the entity.

Validation formulae can be attached to any entity and since slots themselves are also modeled as entities, to any slots as well. These formulae may customize the refinement rules of the given entity. The formulae are specified as DMLA operations using the built-in operation language. The operation language is also completely modeled.

A constraint defines a reusable validation mechanism (e.g. type conformance check) attachable to slots. Refinement of slots is usually driven by adding new constraints and further down narrowing existing ones, thus restricting the structure and/or the behavior of the slot. DMLA offers built-in constraints for re-occuring tasks and allows creating universal, or domain-specific constraints in a flexible way. The two most important built-in constraints kinds are: i) type constraint restricting the type of the values to be put in the slot (e.g. when filling the slot *Frame* in entity *RaceBike*, one can only use refinements of *RaceFrame* entity there), and the ii) cardinality constraint prescribing the allowed number of refinements within a given slot (e.g. *Configuration* may have zero-to-many *Component*s).

Although the basic, built-in entities are not detailed in this paper, one of them, *ComplexEntity*, should be mentioned since it acts as a base entity for domain modeling. It has a slot called *Fields*. The cardinality of the slot *Fields* allows any number of refinements (0..*) of any practically available type. Since all slots in DMLA must have a meta, it is not possible to add new features to an entity, unless the meta-entity has an appropriate meta-slot. The *Fields* slot allows one to extend entities with new slots. By omitting the slot, refinements can deny the ability to introduce any further slots.

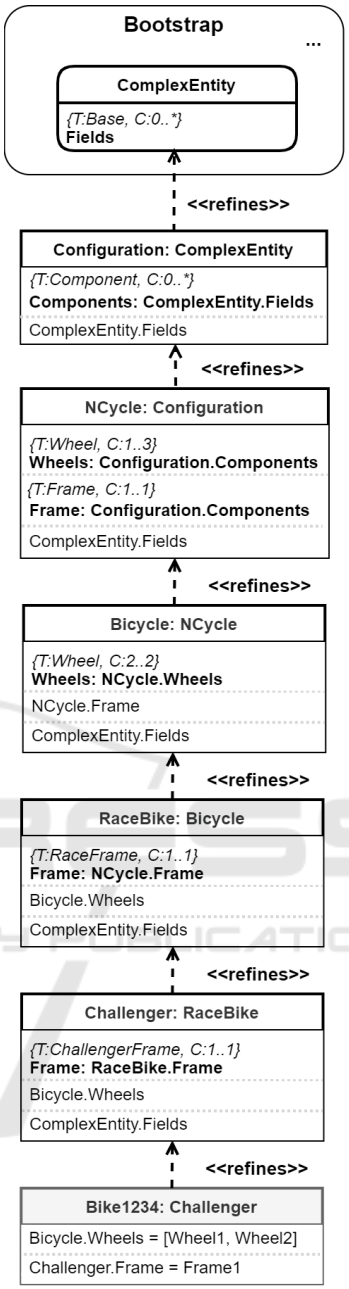

Figure 1: DMLA model fragment.

Figure 1 shows a DMLA model fragment (an example refinement chain), depicting entities as boxes. This model fragment is also used in Section 4 and 5 to illustrate our level-spanning model transformation concepts. *ComplexEntity* is represented by a rounded rectangle indicating that it is not part of the domain model, while concrete objects are differentiated from more abstract entities by their gray color. The refinement relationship between the entities are denoted by

dashed arrows with "*refines*" stereotypes. For example, *RaceBike* is a refinement of *Bicycle*.

The slots are shown embedded into entities similarly to attributes of a class in UML class diagrams. Attributes pointing to another entity are modeled by slots and therefore visualized embedded in the box, not as associations between the entities. Meta-slot relationships are represented by *Slot: MetaSlot* labels. In DMLA, slots can be explicitly kept (cloned) or omitted. Whenever a new slot is introduced, it is shown in bold (e.g. introducing slot *Wheels* in *NCycle*). Constraints of slots are denoted above the corresponding slot in curly brackets with *T:* (type constraint) and *C:* (cardinality constraint).

The entry point of domain definition is the *ComplexEntity*. *ComplexEntity* uses a different notation (rounded rectangle) compared to other entities, as it is not part of the domain definition. This refinement chain in the model exemplifies a key feature of the underlying DMLA: step-wise refinement. One starts from a highly abstract entity and refines it step-by-step until a fully concretized object is eventually obtained. For example, *RaceBike* references supported frame type via a slot (*Frame*). When refining *RaceBike* and creating the *Challenger* entity, the type constraints applied on slot *Frame* are narrowed so that the slot can contain only refinements of the *Challenger-Frame* entity. Type constraints automatically ensure that the concretization is always consistent whenever the validation succeeds, thus there is no need to define additional constraints.

4 MOTIVATION

In the scope of mainstream two-level modeling, Model2Model transformation aims to provide a mean to specify the way to produce target models from a number of source models. For this purpose, it should enable developers to define the way source model elements must be matched and navigated in order to initialize the target model elements. Formally, a simple model transformation has to define the way for generating a model M_2, conforming to a meta-model MM_2, from a model M_1 conforming to a meta-model MM_1.

Figure 2 summarizes a typical model transformation process in a standard modeling scenario. A model M_1, conforming to a meta-model MM_1, is transformed into a model M_2 that conforms to a meta-model MM_2. The transformation is defined by a model transformation language (such as ATL (Jouault et al., 2008) or ETL (Kolovos et al., 2008)) that enables to specify how one (or more) target model can be produced from a set of source model. The MM_1

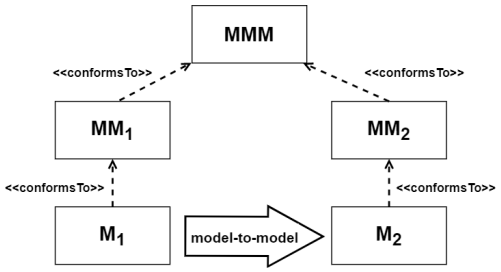

Figure 2: Classic model transformation approach.

and MM_2 meta-models, has to conform to a meta-meta-model *MMM* (such as Ecore).

In the multi-layer setup of DMLA, however, it cannot be presumed that all elements of the model transformation pattern are on the same abstraction level, like the modeling elements of M_1 and M_2, because there can be more than two meta-levels at a time and meta-levels can also influence other levels beyond the immediate one. For example, in the refinement chain of figure 1 *Configuration* is refined five times before it reaches its fully concretized state as *Bike1234* and the contained slots gain more and more concrete information as we getting more and more concrete along the refinement chain. Moreover, in early phases of modeling at higher abstraction levels one may not know the concrete name and the structure of the modeling elements of lower levels for which the transformation rule should be formulated. For example, if one may want to formulate a transformation rule at the level of *Configuration*, extra conditions need to be added in order to match the desired source model elements at lower levels.

In the next section, we introduce our work-in-progress transformation language which can be an ideal candidate to capture level-spanning model elements in transformation rules across multiple abstraction levels.

5 THE TRANSFORMATION LANGUAGE

By designing DMLA Transformation Language (DTL), our primary goal is to provide a flexible and concise way for the model-to-model transformation of level-blind multi-layer models in DMLA. DTL is very similar to ATL (Jouault et al., 2008) in that in both languages transformation code is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models. Each transformation rule contains a unique name. It is introduced by the keyword rule that is followed by the rule's name. Its logic

is surrounded by curly brackets. In the source pattern (*from* part), rules declare which element type of the source model has to be transformed, while in the target pattern (the *to* part), rules declare to which element(s) of the target model the source pattern has to be transformed to. The implementation of the target pattern declares the details of the transformation.

DTL is a hybrid of declarative and imperative style. The basic style of transformation writing is declarative, which means mappings and matching rules can be easily expressed in the source pattern. However, imperative constructs are provided in the target pattern so that basic operations like add and delete can also be expressed.

One of the most important features of DTL is that it allows to customize the level-spanning model elements to search in the refinement chain. For each model element (entity, slot, constraint) one can specify additional conditions to restrict what is included in the source model. It also possible to specify the searching strategy for refinements: (i) inclusive transitive refinements with **:|** operator, (ii) exclusive transitive refinements with **:^** operator and (iii) direct refinement with **:** operator. The term "inclusive transitive refinements" covers the matching modeling element and all of its refinements, while the term "exclusive transitive refinements" excludes the matching modeling element and only transitive refinements are included. "Direct refinement" refers to the immediate refinement of the matching modeling element.

In the following, we explain some of the features of DTL through practical transformation rules formulated for the domain model of Figure 1. For the sake of clarity, we have slightly simplified the model fragment of Figure 1, thus only those parts of the entities are displayed that are necessary to illustrate the mechanisms of DTL. We use fictional requirements to which we formulate transformation rules in order the modify and refactor the original DMLA model fragment.

REQ1: *Each bicycle model has a regular sales price.*

Listing 1 depicts the transformation rule for adding the *Price* slot to the inclusive refinements of *Bicycle* entity. The slot is originated from *ComplexEntity.Fields*, it has a *Number* type and zero-to-one cardinality . *Price* slot is added with an imperative statement in the target pattern. Note that is also possible to specify identifiers for the matching elements in the source pattern, e.g. identifier *bicycle_s* refers to the matching inclusive refinements of *Bicycle* entity.

```
rule AddPriceSlot
{
  from {
      entity bicycle_s:| Bicycle {}
  }

  to {
    @Type: Number
    @Cardinality: 0..1
    new slot Price: ComplexEntity.
    Fields;

    add Price to bicycle_s;
  }
}
```

Listing 1: Add Price slot to Bicycle.

REQ2: *The introduction of new features is no longer allowed in Bicycle refinements.*

According to this requirement, we should deny the introduction of new features by omitting *ComplexEntity.Fields* and its direct slots. By omitting these slots, refinements deny the ability to introduce any further slots. Note that we also formulate additional conditions for the matching slot: only slots that have *Base* as their type constraint will be matched. Listing 2 depicts the transformation rule for deleting direct *Fields* elements from inclusive transitive *Bicycle* refinements.

```
rule DeleteDirectFieldsElements
{
  from {
    entity bicycle_s:| Bicycle
    {
      slot matchingSlot:
       ComplexEntity.Fields {
        constraint t1:| TypeConstraint
        { slot .Type = Base  }
  }}}

  to { delete bicycle_s.matchingSlot; }
}
```

Listing 2: Delete direct Fields elements.

REQ3: *Race bike models are no longer supported. From now on, Challengers can be considered Bicycle models.*

Listing 3 depicts the transformation rule for changing the meta of *Challenger* to *Bicycle*. We search *challenger* as the direct refinement of *RaceBike* and *bicycle* as the direct refinement of *Configuration*. Then, meta of *challenger* is set to *bicycle*.

```
rule ModifyMetaOfChallenger
{
  from {
    entity challenger: RaceBike   {}
    entity bicycle: Configuration {}
  }
  to { challenger.meta = bicycle; }
}
```

Listing 3: Change meta of Challenger.

6 CONCLUSIONS

In this paper, we presented DTL, our work-in-progress multi-layer transformation language, highlighting its features regarding capturing level-spanning model elements. Although the paper focused on the level-blind setup provided by DMLA, we believe that our experiences and conclusions are worthy of general discussion. Even though the DTL language is in explanatory phase, we have presented simplified transformation rules to illustrate the foreseen language mechanisms. Currently, we are developing an ANTLR-based implementation of DTL and also working on more complex case studies in order to demonstrate the feasibility of our level-spanning transformation approach. From a technological perspective, the significance of our research lies in the usage of a modeling language which is based on a refinement relation and not on the classical object-oriented specialization or instantiation relations. Thus,DTL can capture domain concepts across multiple abstraction levels, where the elements can refer to each other independently of their abstraction level. We believe that apart from the technological concepts of DTL, future research should certainly further examine whether DMLA models could be formally represented by multi-level typed graphs, whose manipulation and transformation could be carried out by multi-level typed graph transformation rules.

ACKNOWLEDGEMENT

The work presented in this paper has been carried out in the frame of project no. 2019-1.1.1-PIACI-KFI-2019-00263, which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.1. funding scheme.

REFERENCES

Atkinson, C. (1997). Meta-modeling for distributed object environments. In *Enterprise Distributed Object Computing*, pages 90–101. IEEE.

Atkinson, C. and Gerbig, R. (2016). Flexible deep modeling with melanee. In *Modellierung 2016 - Workshopband : Tagung vom 02. März - 04. März 2016 Karlsruhe, MOD 2016*, volume 255, pages 117–121, Bonn. Köllen.

Atkinson, C., Gerbig, R., and Kühne, T. (2014). Comparing multi-level modeling approaches. In *CEUR Workshop Proceedings*, volume 1286.

Atkinson, C., Gerbig, R., and Tunjic, C. (2013). Enhancing classic transformation languages to support multi-level modeling. *Software and Systems Modeling*, 14.

Atkinson, C. and Kühne, T. (2001). The essence of multilevel metamodeling. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 19–33, Berlin, Heidelberg. Springer-Verlag.

Atkinson, C. and Kühne, T. (2008). Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359.

Coad, P. (1992). Object-oriented patterns. *Communications of the ACM*, 35(9):152–159.

de Lara, J. and Guerra, E. (2010). Deep meta-modelling with metadepth. In *Objects, Models, Components, Patterns*, pages 1–20, Berlin, Heidelberg. Springer Berlin Heidelberg.

Frank, U. (2014). Multilevel modeling: Toward a new paradigm of conceptual modeling and information systems design. *Business & Information Systems Engineering*, 6(6):319–337.

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). Atl: A model transformation tool. *Science of Computer Programming*, 72(1):31–39. Special Issue on Second issue of experimental software and toolkits (EST).

Kolovos, D. S., Paige, R. F., and Polack, F. A. C. (2008). The epsilon transformation language. In Vallecillo, A., Gray, J., and Pierantonio, A., editors, *Theory and Practice of Model Transformations*, pages 46–60, Berlin, Heidelberg. Springer Berlin Heidelberg.

Lara, J., Guerra, E., and Sánchez Cuadrado, J. (2013). Model-driven engineering with domain-specific meta-modelling languages. *Software and Systems Modeling*, 14.

Macías, F., Rutle, A., Stolz, V., Rodríguez-Echeverría, R., and Wolter, U. (2018). An approach to flexible multilevel modelling. *Enterprise Modelling and Information Systems Architectures*, 13:10:1–10:35.

MOF (2005). OMG: MetaObject Facility. http://www.omg.org/mof/. Accessed:2020-03-10.

Sendall, S. and Kozaczynski, W. (2003). Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20:42–45.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.

Urbán, D., Mezei, G., and Theisz, Z. (2017). Formalism for static aspects of dynamic metamodeling. *Periodica Polytechnica Electrical Engineering and Computer Science*, 61(1):34–47.

Urbán, D., Theisz, Z., and Mezei, G. (2018). Self-describing operations for multi-level meta-modeling. In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, pages 519–527. INSTICC, SciTePress.

Wolter, U., Macías, F., and Rutle, A. (2020). Multilevel typed graph transformations. pages 163–182. Springer International Publishing.