VALIDATED VISUAL MODELING OF MULTILAYERS IN DMLA

Sándor Bácsi¹ and Gergely Mezei¹

¹Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Magyar tudósok krt. 2., H-1117 Budapest, Hungary

ABSTRACT

Over the past decade, a number of approaches have been published in the literature, which try to elaborate and clarify the principles of multi-level metamodeling. In this field, however, there are still no widely accepted principles, every approach supports different features. In this paper, we present the basic concepts of a rudimentary prototype which was developed for the purpose of supporting the process of multi-level metamodeling within Dynamic Multi-Layer Algebra (DMLA). We introduce some visualization patterns that can be well-matched to the abstraction levels appearing in the specification, therefore the advantages of multi-level metamodeling can be exploited. The patterns can make it more comfortable and seamless to edit the models. We also summarize the main concerns regarding the validation issues and the visualization of DMLA.

1 INTRODUCTION

In the software industry, it is a well-known fact that knowing exactly what the system must do is at least halfway down the road towards success. Obviously, customers try to express their expectations at their best, but these initial requirements may change during development. This behavior is not due to the lack of competency of software engineers or the customer at all. It is indeed a genuine consequence of software engineering. In the beginning of a complex project, it is hardly possible to capture all requirements exactly and to meet all design challenges in advance.

Software development usually begins with a few design sketches capturing only major project requirements. At this stage, the software model is very flexible, almost an empty template containing lots of implementation gaps. As development continues, uncertainty is shrinking and the size of the gaps is being reduced as the details are converging towards their final form. It is essential to support this iterative, evolutionary behavior by model-based software development methods in order to effectively apply modeling in development. However, most of the current metamodeling approaches take it for granted that the meta-model does not evolve after a short and intense introductory period.

One of the new branches of metamodeling focus on multi-level metamodels. By increasing the number of modeling levels, we can obtain an environment, where the definition of domain concepts are composed of fine graded steps simplifying thus the customization of the domain. However, there is no consensus in the literature on the exact meaning of the methodology and there are no widely accepted principles, which makes practical application hard.

DOI: 10.26649/musci.2019.017
In recent years, a wide range of tools and methodologies have been introduced in the field of multi-level meta-modeling. One of these approaches is the Dynamic Multi-Layer Algebra (DMLA), which is a flexible, self-validation and formal multi-level modeling framework. In DMLA, all model elements are stored as 4-tuples and all operations are applied on these tuples. DMLA incorporates a fully self-modeled textual operation language (DMLAScript) above the 4-tuple representation, which provides a user-friendly interface to reach and manipulate the tuples. Due to the initial concepts and the textual representation, DMLAScript has its drawbacks in the effective manipulation of the domain model, especially when focusing on realistic, evolutionary model editing. In this paper, we present a prototype which was developed for the purpose of supporting the process of DMLA. We introduce some visualization patterns that can be well-matched to the abstraction levels appearing in the specification, therefore the advantages of multi-level metamodeling can be exploited. The patterns can make it more comfortable and seamless to edit the models. We also summarize the main concerns regarding the validation issues and the visualization of DMLA.

The paper is organized as follows: Section 2 presents the background and the related work. Section 3 shortly introduces the DMLA approach in order to give clear and precise understanding of the main concepts. Section 4 presents the improved version of the prototype visual DSL in brief, while visualization patterns are introduced in Section 5. The main concerns regarding the validation are summarized in Section 6. Concluding remarks are outlined in Section 7.

2 RELATED WORK

In this section, we summarize the most relevant approaches in the field of multi-level metamodeling. On the other hand, we also summarize existing visualization approaches regarding multi-level metamodeling. OMG's Meta-Object Facility [1] (MOF) is often referred to as the de-facto standard for implementing metamodel based solutions. MOF provides a four-layer architecture, which can be satisfying for most of the problems. However, alternative as multi-level solutions pointed out [2], it is not always enough neither in flexibility nor focusing on the preciosity. To overcome the weaknesses of MOF, n-level metamodeling has gained increasing popularity in the last decade.

Here, it is worth to explicitly differentiate between linguistic and ontological metamodeling based upon separate linguistic and ontological instance-of relations.

Several approaches also differentiate between shallow and deep instantiation. Shallow instantiation means that the domain information is available exactly on one modeling layer below its definition. In opposition to shallow instantiation, deep instantiation means that the domain information may be used at layers below as well according to its parametrization.

One of the most relevant deep instantiation techniques is the so-called potency notion [3]. Here, a potency is assigned to each model entities. The assigned potency value represents the number of model levels the certain element can get through before reaching its fully instantiated state. Melanee [4] is a deep modeling tool based upon the concept of orthogonal classification and potency notion.
Another remarkable deep modeling methodology is the Lazy Initialization Multilayered Modeling Framework [5]. This framework provides an object-oriented modeling language, which is based on lazy instantiation. Lazy initialization can facilitate to manage the instance-of relationships between the layers and their classification.

The metaDepth [6] and XModeler [7] are also multi-level metamodeling approaches. Besides the modeling structure, they provide an operation language as well, which allows the creation of operations within the multi-level metamodeling workbench. In metaDepth, both Java and EOL [8] can be used to specify actions and constraints. XModeler provides an executable programming language, XOCL [9], which is based on OCL.

Recently, the demand for an appropriate visualization has arisen also in the field of multi-level metamodeling. XModeler provides a visual editor to support the process of multi-level modeling. As a concrete syntax, there is a named box notation in XModeler's visual editor, which represents the class of the given entity. Hence, it is possible to navigate among instances and class declarations. XModeler has a color node notation as well in order to be able to distinguish the levels of type abstraction. The header of each class has a specified background color to express the appropriate level of type abstraction.

Melanee provides a diagrammatic workbench to facilitate the process of deep-modeling. The Melanee workbench uses clabject notations and provides a wide range of tools to build a deep-model. The main advantage of the Melanee workbench is the dynamic display of the currently usable types. The deep-model can be edited by using both the textual and the visual editor.

In our former paper [10], we pointed out the drawbacks of DMLAScript categorized into two groups: (i) The management of the relations among entities; (ii) Refactoring and editing of certain entities. We introduced different views, which can facilitate the process of multi-level metamodeling on a higher abstraction level. We presented a simple prototype with a limited set of features. In this paper, we introduce an improved version of the prototype with the related visualization patterns.

3 DMLA IN A NUTSHELL

Dynamic Multi-Layer Algebra [11] is our multi-level modeling framework that consists of two parts: (i) the Core, containing the formal definition of modeling structures and its management functions; (ii) the Bootstrap, consisting of a set of essential entities that can be reused in all domains. In DMLA, the model is represented as a Labeled Directed Graph, where all model elements have four labels: i) the unique ID of the element, ii) a reference to its meta element by its unique ID, iii) a list of concrete values, and iv) a list of contained attributes. Besides the 4-tuples representing the model entities, there exist functions that manipulate the model graph, thus forming the Core of DMLA, which is defined over an Abstract State Machine (ASM) [12]. The states of the state machine represent the snapshots of dynamically evolving models, while transitions (e.g., deleting a node) stand for modifications between those states. The Bootstrap extends the Core by making it more usable in practice. The Bootstrap is an initial set of modeling constructs and built-in model
elements (e.g., primitive types) that are needed to adapt the abstract modeling structure to practical applications when we build our domain models. It is worth noting that the separation of the Core and the Bootstrap allows the creation of several different Bootstraps (defining different meta-modeling paradigms), but so far, we have created one standard Bootstrap that fits our research goals.

Instantiation in DMLA has several specialties. Whenever a model entity claims another entity to be its meta, the framework automatically validates if there is indeed a valid instantiation between the two entities. However, unlike other modeling approaches, the rules of valid instantiation is not encoded in an external programming language (e.g. Java), instead, it is modeled by the Bootstrap. All validation formulae can be modularized by being introduced directly into the Bootstrap. This even applies to constraints, like checking type and cardinality conformance. Since the validation formulae directly influence the proper semantics of instantiation, the instantiation is self-defined via the model per se. The technical facility enabling this self-described meta-modeling is based on operation reification. Operation definitions are modeled by their abstract syntax tree (AST) representation as tuples, which are later translated into executable code by the framework. In DMLA, multi-level behavior is supported by fluid metamodelling, meaning that it is not required to instantiate all entities of a model at once. Hence, in DMLA, instantiation steps are independent by design. Each entity can refer to any other entity along the meta-hierarchy, unless cross-level referencing is found to be contradictory during model validation. Entities may have attributes referred to as slots, describing a part of the entity, similarly to classes having properties in object-oriented programming. The concept of slots is modeled in the Bootstrap.

4 MAIN CONCEPTS OF THE PROTOTYPE

We elaborated a visual DSL using the Eclipse Modeling Framework [13] and Sirius [14] to visualize the meta-hierarchy by constructing a tree of the model showing the instance-of hierarchy. We used EMF to define the abstract syntax of the visual DSL and we applied the features of Sirius to simplify the graphical representations and highlight the elements of interest.

Let us summarize the main concepts of the modeling process using the textual DMLAScript. Entities may have attributes referred to as slots, describing a part of the entity, similarly to classes having properties in object-oriented programming. The concept of slots is modeled in the Bootstrap. As an example (based on our textual DMLAScript solution for the Bicycle Challenge [15]), an entity Bicycle may have slots for its Fork, Seat and Frame components. Each slot originates from a meta-slot defining the constraints to obey to. When instantiating the entity, all of its slots are validated against the meta-slots. This is where we check the modeled type and cardinality constraints of the slot, along with other applied constraints. In our previous example, the type constraint applied to the slot Seat restricts the value to be an instance of the entity Seat.

Previously, the initial prototype of the visual DSL only supported a rectangular box notations for each entity with the Entity:MetaEntity and Slot:MetaSlot relations.
In the improved prototype, the constraints can be specified on the slots with the following concrete syntax:

\[
\text{Slot: Metaslot \{T: \langle TypeConstraint \rangle, } \\
\text{C: \langle CardinalityConstraint \rangle, } \\
\text{OP: \langle OperationSignature \rangle\}}
\]

Here, it is worth to emphasis that optional-mandatory slots are modeled by cardinality 0..1 and the so-called MustFillOnce constraints at the same time. MFO notation can be used for MustFillOnce constraints. OP notation is used for the OperationSignature constraint. The OperationSignature constraint delivers semantics that is similar to its usual programming language peers.

5 VISUALIZATION PATTERNS

In this section, we introduce a set of visualization patterns, which can be well-matched to the process of multi-level modelling:

**Visualization of the meta-hierarchy:** In design-time, it can be essential to navigate up or down on the relevant part of the hierarchy in order to get more information about a certain entity. In a visual representation, it is possible navigating through the meta-hierarchy in a seamless way, therefore the modeling process can be more comfortable in design-time. Fig. 1 shows a visualized meta-hierarchy as an example from the Bicycle Challenge case study.

![Figure 1: A visualized example from the Bicycle Challenge case study](image)

**Explicit Omission:** When new features are needed, the domain modeler can create new slots by dividing general purpose slots. If adding new features may be required later, the original slot can be kept or it can be omitted otherwise. In the visual workbench, the explicit omission can be enabled on each slot with an isOmitted flag.

**Constraints:** Mandatory slots are modeled by cardinality 1..1. They must be kept all along the whole instantiation chain. Optional slots are modeled by cardinality 0..1. They can be omitted on any level. Optional-mandatory slots are modeled by cardinality 0..1 and the MustFillOnce constraints at the same time. They can be omitted at any level as far as their value has already been set earlier along the
instantiation hierarchy. Fig. 2 shows an example to demonstrate the different kinds of cardinality constraints in DMLA.

Figure 2: Cardinality constraints example

**Concrete objects:** While getting more and more concrete along the instantiation chain, the slots gain more and more concrete information. It has to be visualized what entities are concrete objects in order to be able to differentiate them from non-concrete entities. Fig. 3 shows *MyConcreteBike* entity as a concrete object.

Figure 3: *MyConcreteBike* as a concrete object

### 6 VALIDATION ISSUES

Code generation and validation logic are not supported in the presented prototype yet. The code generator shall convert the current modeling structure to DMLA’s 4-tuple representation. However, the question of the validation is not that simple here as it seems at first. The EMF model stores the abstract syntax definition of the visual DSL, and currently there are some validation rules which can validate the EMF model, but this is not the native validation of the DMLA model above the 4-tuple representation. Validation in DMLA is intuitive: whenever a model entity
claims another entity to be its meta, the framework automatically validates if there is
indeed a valid instantiation between the two entities. All validation formulae can be
modularized by introducing directly into the Bootstrap. On the other hand, the
validation is also based on three types of formulae: alpha, beta and gamma. These are
the main reasons why the validation should be done in DMLA, and the results of the
validation should be bound to the EMF model. In this way, we can easily highlight
the validation errors in the Sirius representation.

For the purpose of binding, we decided to use the fully qualified name of the
DMLA model elements. A fully qualified name is an unambiguous name that
specifies which object, function, or variable a call refers to without regard to the
context of the call. In DMLA’s hierarchical structure, a name is fully qualified when
it is complete in the sense that it includes all names in the hierarchic sequence above
the given element and the name of the given element itself.

The structure of the EMF entities are somewhat similar to a hierarchical
structure, therefore the fully qualified name of the DMLA elements can be well-
matched. If an error happens during the native validation of DMLA, the framework
shall highlight the error in the Sirius representation, therefore it can be easily
recognized where to search for the given error.

7 CONCLUSIONS

Traditional modeling approaches have difficulties in following the iterative,
evolutionary behavior of software projects. It is clear that these traditional approaches
do not support the step-wise refinement of requirements in a natural way. The demand
for supporting dynamism at all levels of modeling has been attracting much more
attention than a few decades ago. The real challenge in supporting this trend is to
grant flexibility, but ensure validity at all time. DMLA can provide flexible, dynamic
and formal modeling approach. This paper has discussed some basic visualization
patterns which can be well-matched to the abstraction levels appearing in the
specification, therefore the advantages of multi-level metamodeling can be exploited.
The patterns can make it more comfortable and seamless to follow the process of the
step-wise refinement.

In the future, we aim to add the the code generation and validation support to
the rudimentary prototype which we introduced in this paper. We would like to
introduce a set of views to the prototype, which we have already introduced in our
former paper. We plan to investigate our visualization concept in the context of other
relevant multi-level metamodeling approaches by creating a comparison between the
proposed method and relevant existing methods in a visual way. We are also working
on new case studies, which can be tested in the context of this visualization approach.

ACKNOWLEDGEMENTS

The research has been supported by the European Union, co-financed by the
European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental
Research Collaborations Grounding Innovation in Informatics and
Infocommunications).
BIBLIOGRAPHY


