The Lazy Initialization Multilayered Modeling Framework (NIER Track)

Fahad R. Golra
Université Européenne de Bretagne
Institut Télécom / Télécom Bretagne
fahad.golra@telecom-bretagne.eu

Fabien Dagnat
Université Européenne de Bretagne
Institut Télécom / Télécom Bretagne
fabien.dagnat@telecom-bretagne.eu

ABSTRACT

Lazy Initialization Multilayer Modeling (LIMM) is an object oriented modeling language targeted to the declarative definition of Domain Specific Languages (DSLs) for Model Driven Engineering. It focuses on the precise definition of modeling frameworks spanning over multiple layers. In particular, it follows a two dimensional architecture instead of the linear architecture followed by many other modeling frameworks. The novelty of our approach is to use lazy initialization for the definition of mapping between different modeling abstractions, within and across multiple layers, hence providing the basis for exploiting the potential of metamodeling.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design tools and techniques; D.1.5 [**Programming Techniques**]: Object-Oriented Programming; H.1 [**Information Systems**]: Models and principles

General Terms

Design, Standardization

Keywords

Metamodeling, Strict metamodeling, Instantiation, LIMM

1. INTRODUCTION

Modeling was once considered a tool to understand the System Under Study (SUS) but recently it has emerged out as a key activity of system development. This led to the rapid and incremental development of UML. However UML restricts modeling to 1) a 4 layers hierarchy, 2) 'strict' metamodeling (a model element must be an instance of exactly one meta-model element), and 3) the exclusive use of *instance-of* relationship amongst adjacent modeling layers. Holding to backward compatibility, UML2.0 could not unleash the complete potential of metamodeling. Different researchers have been using its evolution mechanism (UML Profiles) to add up to its capabilities [6]. Some recent approaches of metamodeling (deep instantiation, power type based modeling, etc.) question its shortcomings and present their own methodologies [2, 1]. However none of these approaches succeeded in replacing or influencing UML.

In order to exploit the potential of metamodeling both at micro level (inside a model) and macro level (between models), LIMM is presented as an object oriented multilayered modeling language that aims to provide a better support for the definition of different DSLs. The usage of lazy initialization in modeling is a novel idea that adds a lot of flexibility for the modelers, without introducing any functional limitations. In our proposal, the designer of a model can specify for each model element if it is at meta level, at application level or it is a data. To achieve this goal, flags are associated to model elements to control the way they can be used in the subsequent layers. A flag can take three different values to *restrict*, *allow* or *force* the initialization of a model element in the next adjacent layer.

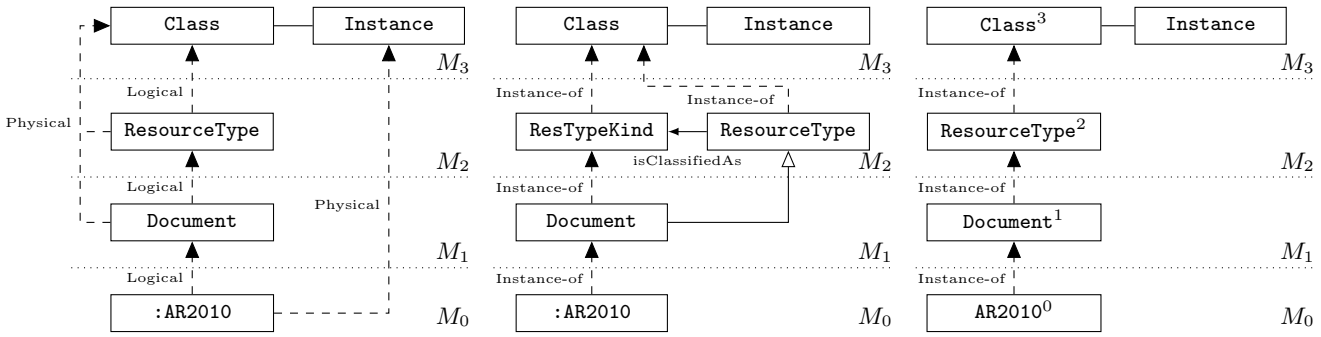
This paper is structured as follows. Section 2 explains the multilayered modeling framework and describes the recent associated endeavors. Section 3 describes the LIMM framework. Section 4 presents the metamodel for LIMM framework. Finally Section 5 outlines the conclusions and future work.

2. MULTILAYERED MODELING

When modeling started to mature, modeling languages themselves needed to be modeled (language-ware). This led to a shift from traditional bi-layered modeling towards multilayered modeling hierarchy. UML modeling framework as defined in MOF rests on a 4 layers hierarchy. Though UML holds the industry, still some modelers (specially the meta-CASE tool designers) face some problems using it. The well known problems of this framework are *Ambiguous Classification*, *Replication of Concepts*, *Structure redundancies* and *problems with metalevels* [4, 1, 2, 7].

One of most discussed problems in literature regarding the UML modeling framework is its incompetence to differentiate between the logical and the physical flavors of the *instance-of* relationship. As illustrated in Figure 1(a), the document `AR2010` is an *instance-of* class `Document`, which is an instance of both `ResourceType` and `Class`, where the two instantiations are not the same. The first instantiation relationship is a *logical* relationship mapping an instance to a concept of the system under study (SUS), whereas the sec-

ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA



(a) UML (b) Power Types (c) Deep Instantiation

Figure 1: Comparative Example in different Multilayered modeling frameworks

ond relationship indicates its *physical* existence as a class. Thus the concept of object/type facets is addressed in MOF, but is not very well defined. The term *clabject* coined by Atkinson refers to this concept [1].

A unique notion of *instance-of* relationship between two layers seems unable to portray the concept of class as an instance. Power type based modeling proposed to use instantiation to carry on the instance facet to the subsequent modeling layer, and inheritance to carry the class facet [2]. In this framework, a class (such as `Class` in Figure 1(b)) has two instances in the subsequent modeling layer. First, the power type, `ResTypeKind`, carries the object facet to the next level using the *instance-of* relationship. Second, the partitioned type, `ResourceType`, assumes the class facet and is intended to be the super class of the power type's instance (`Document` in the example). Instead of having a simple type-instance relationship like in traditional modeling, it creates a triplet of the power type, the partitioned type and the class at a lower modeling layer, as depicted in Figure 1(b).

The drawback of power type based modeling is its inability to define associations in terms of clabjects. All the complexity of partitioning the power type from partitioned type in order to classify the different facets is transferred to the semantics of the relationship *isClassifiedAs*, which lacks concrete explanation. Furthermore, the inheritance relationship across different layers in the architecture makes it quite complex to draw the boundaries of a modeling layer.

In contrast to power type based modeling, deep instantiation based modeling consents to defer the instantiation for a "known" number of layers [1, 4]. Deep instantiation uses the concept of potency, where each association, attribute and class is assigned a potency. The potency amounts to the number of levels an artifact can deny instantiation. Thus if an attribute is assigned a potency of 2, and it resides at layer M_n, then it would not be instantiated till M_{n-2}. As it gets down to M_{n-2}, it will get instantiated and become a slot. Deep instantiation addresses the dual nature (class and instance facets) of the classes using this potency. An example of deep instantiation can be viewed in Figure 1(c).

One of the major drawbacks of deep instantiation is that one has to know in advance, the number of layers each artifact should resist instantiation. The metamodeler must anticipate the uses of its metamodel and fix the potency of its elements. This fixed potency eventually takes the flexibility out of the reach of the application modeler, who is constrained to follow it.

Álvarez et al. suggested that instead of a linear modeling hierarchy, one should follow the nested hierarchy [5]. In such a hierarchy, the level M_0 is adjacent to both M_1 and M_2 layers. So the instance at level M_0 can have *instance-of* relationships both from M_1 and M_2. The essence of nested layers is that it is recursive in nature and has a conforming mapping structure called the *G-mapping*. The drawback of using such a multi layer structure is that, though it works fine for 3 layers, but as it progresses to more layers, the original benefit being sought is lost. Thus it is restricted to a maximum of 4 layers. And the problem of *ambiguous classification* still remains; as the only relationship allowed is *instance-of* and no well defined differentiation among the different flavors of *instance-of* relationship is discussed.

A detailed study of the existing multilayered modeling frameworks shows that each of these techniques tries to resolve a part of the explored shortcomings of UML. None of these techniques, other than UML, gained popularity in the software industry because of the respective drawbacks detailed above. For these reasons, we propose a general multilayered modeling framework influenced by the previously presented approaches but relying on a lazy initialization mechanism inspired from functional programming.

3. LAZY INITIALIZATION

The Lazy Initialization Multilayer Modeling (LIMM) takes its inspiration from lazy functional programming, where evaluation may be delayed until required. Using lazy initialization helps clarify the *instance-of* relationships between the layers and their classification.

In UML, the *links* in the data models are the instances of the *associations* in the model at upper layer; same goes for the *slots* which are the instances of the *attributes* in the model at upper layer. But the *instance-of* relationship which bypasses the layers is not well explained. LIMM uses a special sort of abstraction known as *interconnection* to specify the different relationships that bypass the model boundaries. This framework gives a proper justification and sound basis to various relationships within a model or amongst different models, residing either on same or different abstraction levels.

We argue against deep instantiation because one cannot decide before hand, the number of layers after which a model element is to be initialized. Indeed, we believe that the number of layers cannot be fixed and must result from the choice of the designer. Furthermore, each layer is usually built by a different designer, making it impossible to predict when a

model element will reach its object layer (where it is initialized). Our proposal is that at a certain layer, the designer must specify the initialization scheme of model elements in the subsequent layer. The initialization scheme can either forbid the initialization or devolve the decision, to choose the number of layers before initialization, to the designer of the subsequent layer.

The flag value of the associated model element specifies whether it is going to be initialized in the subsequent layer or not. This flag can be null or have three different values: 2 for *Restrict*, 1 for *Relax* or 0 for *Force*. A null flag describes an instantiated model element. While developing a model, the flags of the meta-elements in the reference model are tested to determine which elements must or may be initialized at current layer. A flag value 2 (*Restrict*) specifies a model element that cannot be initialized in the subsequent layer (where the flag value can shift to 2, 1 or 0). A flag value 1 (*Relax*) allows the initialization of a model element in the subsequent layer (where its flag value can become 1, 0 or null). Lastly, a flag value 0 (*Force*) forces the designer of the subsequent layer to initialize the model element (moving its flag to null).

Models being model elements maintain their own flags to specify their abstraction level. Models having a flag value 2 are at some metalevels whereas the models having the flag value 0 are the user models. Models having a flag value 1 can be intermediate level models that can/cannot be initialized. A model contains classes and these classes contain attributes. This containment hierarchy constrains flags: the flag value of a container must be greater than or equal to the flag value of its contained elements. Data models or object models do not carry a flag, so they have a null value. The flag values of a model can either be specified by the designer or calculated automatically.

LIMM uses a two dimensional modeling framework with a horizontal division between the core multilayering metamodel and the linear modeling hierarchy, which is further divided into multiple modeling layers. The core multilayering metamodel is the LIMM metamodel that defines the language in which the models placed in the various layers are defined. This is a top level meta-metamodel that defines itself. Our numbering of the linear modeling hierarchy, in contrast to UML, keeps augmenting from the topmost layer, M_0, to the subsequent layers up to M_n.

4. LIMM METAMODEL

LIMM metamodel presents two common models; a *user model* and a *reference model*, which are connected through *interconnections*, as shown in Figure 2. A *reference model* can be referenced by another *reference model* or by a *data model*. A *data model* is a model which is not referenced by any other model. The *core multilayering metamodel* does not have a flag because all other models in its linear hierarchy are *instance-of* this model, thus all the *physical instance-of* relationships are implicit in this model. The dual nature (type/instance) of a model element is described in terms of initialized and non-initialized contained model elements. The *common model* is itself a *model element* to support abstraction hierarchies. All the meta-metamodels for defining new DSLs are presented as an extension of the *core multilayering metamodel*. This allows them to utilize multiple layers for model definitions.

Channels and *Items* are both *model elements*, which are

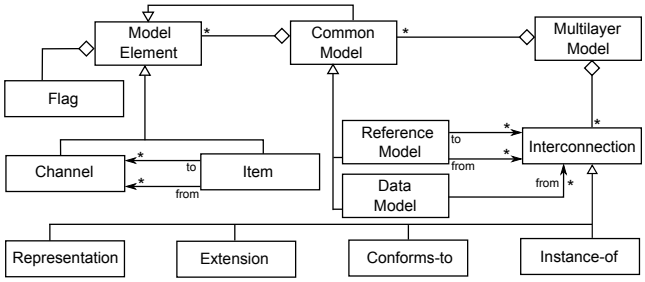


Figure 2: Minimal LIMM metamodel

used to relate a model to the graph semantics. They are kept open to extension for new abstractions in order to support the definition of new DSLs. One of the hallmarks of UML was the visual representation through adding semantics to the graphs. LIMM uses the same methodology but maintains a distinction between the *channel* and *interconnection*. A *channel* is a relationship between two model elements within a model, whereas an *interconnection* is a relationship between elements of different models. Thus the semantics of relationships crossing the boundaries of a model is different from those who do not. The classification of *interconnection* is done in terms of abstraction levels. A *Cross Layer Interconnection* (CLI) relates two model elements residing on different layers, whereas a *Same Layer Interconnection* (SLI) relates the model elements residing on the same layer. *Instance-of* and *Conformity* are CLIs that contravene the strict metamodeling of UML, where the only relationship across layers can be *instance-of*.

A DSL for metamodel specification, Kernel MetaMetaModel (KM3) [3], defines the structure of a model using directed multigraphs. We follow their definition and extend it further for describing our specification. Let $G_M = (N_M, E_M, \Gamma_M)$ be the directed multigraph for the common model M, where

- N_M is the set of nodes (representing the *Items*);
- E_M is the set of edges (representing the *Channels*);
- Γ_M is the function from E_M to $N_M \times N_M$ associating an edge to its extremities.

A *common model* M is represented as $M = (G_M, R, \kappa_M)$, where

- R is the *reference model* of M. The associated graph of this *reference model* is $G_R = (N_R, E_R, \Gamma_R)$.
- κ_M is a function from $N_M \cup E_M$ to $N_R \cup E_R$. It relates the elements of the model M to the elements of the *reference model* R (meta-elements), thus representing the explicit logical relationships.

Having the *common model* and *reference model* in hand, we are able to represent the *multilayer metamodel* $ML = (G_{ML}, CMM, \kappa_{ML})$, such that

- $G_{ML} = (N_{ML}, E_{ML}, \Gamma_{ML})$ is a directed multigraph for the *multilayered model*, where
  - N_{ML} is the set of nodes of the graph representing the elements of *multilayer model*, $N_M \cup N_R$.
  - E_{ML} is the set of edges of the graph representing the associations in *multilayer model*, $E_M \cup E_R$.

– Γ_{ML} is the function from E_{ML} to $N_M \cup E_M \times N_R \cup E_R$ associating this edge to its extremities.

- *CMM* is the *core multilayering metamodel* which defines itself. It is the topmost level of hierarchy, where the meta-metamodel is defined recursively by itself.

- κ_{ML} is a function from $N_{ML} \cup E_{ML}$ to N_{CMM}, relating all the model elements and relationships of the *multilayer model* to the nodes of *core multilayering metamodel* representing the physical relationships. These relationships are implicit in the visual representation.

A *reference model* is referenced by a data model from a lower layer but at the same time it may refer to some other *reference model* (acting itself as *data model*). The elements of the *reference model* R are hence meta-elements for the elements of the models residing in the subsequent modeling layer. The CLIs (instantiation and conformity) can be defined when M and R are both at different layers, whereas SLIs can be defined when M and R are both at the same layers. For a model M, a partial function fo_M gives the flag value of its model elements if it is not null. Thus, $\forall e \in N_M \cup E_M, fo_M(e) \in \{0, 1, 2\}$.

Conforms-to relationship (μ) is defined for LIMM as a Cross Level Interconnection (CLI) when,

$$\mu = \{i \in \kappa_M | dom(i) \subseteq dom(fo_M)\}$$

Instance-of relationship (η) is defined for LIMM as a Cross Level Interconnection (CLI) when,

$$\eta = \{i \in \kappa_M | dom(i) \cap dom(fo_M) = \emptyset\}$$

The example used to compare other multilayered modeling frameworks is continued in Figure 3 where a part of object oriented metamodel lies at M_0 layer. The `ResourceType` class at M_1 layer conforms to the class `Class` at M_0. The flag value of the attribute `name` is not initialized till M_3, whereas `category` is initialized at M_2. In this example, `AR2010` has an *instance-of* relationship to `Document`, which in turn has a *conforms-to* relationship with `ResourceType`. The `usedby` link at M_3 is initialized from the `usedby` association at M_2. The flag values on the models and model elements show that they belong to *meta layer*, *application layer* or *data layer*. The instance facet of the class `Document` can be viewed from `category` (slot), whereas the class facet from `name` (attribute). All the relationships explicitly defined are the logical relationships, whereas the physical relationships are implicit in the visual representation, from each model to the *core multilayering metamodel*, adjacent to all the the layers of the *multilayer model*.

5. CONCLUSION

Different multilayered modeling frameworks have been explored to stress their deficiencies and strengths. We proposed to capture the semantics of a model using graphs, where the semantics of the *instance-of* and *conforms-to* relationships can be clarified. A metamodel for the lazy initialization multilayered modeling framework is presented, where lazy initialization schemes are used to resolve the issues faced by multilayered models. LIMM framework supports the definition of new DSLs that need the usage of multiple abstraction levels. In addition, this novel framework improves the level of expression in multilayered modeling by adding semantics to the mappings across layers, which would in turn facilitate model validations. A vital impact on the designer's approach would be the enhanced control over his model to guide the development, without forcing any unnecessary restrictions to the subsequent layer designers. Having models as model elements helps add semantics to the mappings amongst the models also. This inter-model mapping would have a considerable impact in assisting model management.

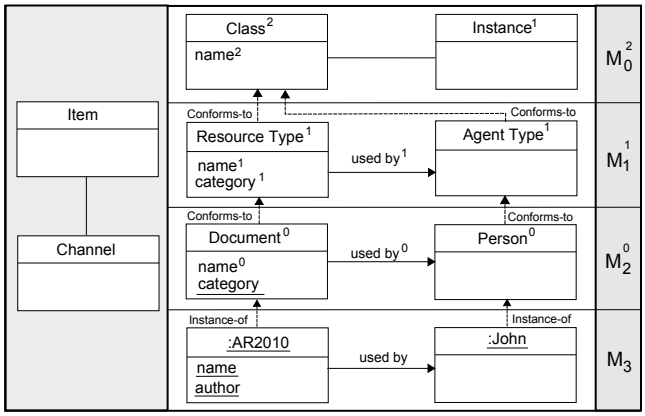


Figure 3: Example in LIMM

The focus of MDE is to exploit domain modeling by improving compatibility and mappings amongst models. We are looking forward to add semantics to the mapping functions between the model elements. This would further assist the automation of software development processes for multi-metamodel applications. For the future work, this multilayered modeling framework would also help to build transformation hierarchies. Having both the model hierarchies and transformation hierarchies would allow LIMM framework to exploit the potential of MDE.

6. REFERENCES

[1] C. Atkinson and T. Kuhne. Rearchitecting the UML Infrastructure. *ACM Transactions on Modeling and Computer Simulation*, 12(4):290–321, 2002.

[2] C. Gonzalez-Perez and B. Henderson-Sellers. A Powertype-based Metamodelling framework. *Software and Systems Modeling*, 5:72–90, 2006.

[3] F. Jouault and J. Bézivin. KM3: A DSL for Metamodel Specification. In *Formal Methods for Open Object-Based Distributed Systems*, volume 4037 of *LNCS*, pages 171–185. Springer Berlin / Heidelberg, May 2006.

[4] T. Kuhne. Understanding Metamodeling. In *Proc. of the 27th International Conference on Software Engineering (ICSE)*, pages 716 – 717, May 2005.

[5] J.Álvarez, A. Evans, and P. Sammut. Mapping between levels in the metamodel architecture. In *Proc. UML - The Unified Modeling Language. Modeling Languages, Concepts and Tools*, pages 34–46. Springer, 2001.

[6] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[7] D. Varró and A. Pataricza. VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (the mathematics of metamodeling is metamodeling mathematics). *Software and Systems Modeling*, 2(3):187–210, October 2003.