Towards rearchitecting meta-models into multi-level models

Fernando Macías[1], Esther Guerra[2], Juan de Lara[2]

[1] Western Norway University of Applied Sciences (Norway)
[2] Universidad Autónoma de Madrid (Spain)

Abstract. Meta-models play a pivotal role in Model-Driven Engineering, as they are used to define the structure of instance models one level below. However, in some scenarios, organizing meta-models and their instances in multi-level models spanning more than two levels yields simpler solutions. This fact has triggered the proposal of different multi-level modelling tools and approaches, although each one of them supports small variations of the multi-level concepts.
In order to benefit from multi-level technology, existing meta-models and their instances could be migrated manually, but this is error prone, costly, and requires expertise for choosing the most appropriate tool and approach. Hence, we propose an automated migration process. This way, starting from a meta-model annotated with multi-level "smells", our approach creates a neutral multi-level representation, and recommends the most appropriate tool according to the required multi-level features. We present an initial prototype, and a preliminary evaluation on the basis of meta-models developed by third parties.

1 Introduction

Modelling in Model-Driven Engineering (MDE) has traditionally adopted a two meta-level approach, where meta-models define the set of admissible models one level below. Instead, multi-level modelling (MLM) [3], also called deep modelling [5], is a modelling proposal that permits the use of an arbitrary number of meta-levels, not necessarily two. This may lead to simpler solutions – with less accidental complexity and a clear specification of classification levels – in situations where the type-object pattern or some of its variants arise [3, 6].

While the dominant practice nowadays follows two-level approaches, our previous studies show that there is a considerable amount of meta-models that could benefit from multi-level technology [6]. In particular, the occurrence of the type-object pattern is common in domains like software architecture or process modelling. For example, in the latter domain, it is frequent the need to model both task types and instances, resource types and instances, agent types and instances, and so on. Using MLM would make such meta-models simpler. However, a manual rearchitecture of a large meta-model into a multi-level version is costly, tedious and error-prone.

Several tools and approaches for MLM have emerged along the years, such as DeepTelos [11], the DPF Workbench [12], Dual Deep Modelling [15], Melanee [1], METADEPTH [5], MultEcore [13], OMLM [9], SLICER [16], XMF [4] and XModeler [7]. Each one of them has its own strengths and limitations, while many implement

small variations of multi-level concepts, like attribute potency or leap potency [8]. Deciding on the tool or approach to use in order to more optimally describe the concepts in a domain can be challenging for novices, and may hamper the adoption of MLM.

To facilitate the migration of standard meta-models into a multi-level setting, we propose automated support for the rearchitecture process and the decision of the most suitable MLM approach given the problem characteristics. For this purpose, first the meta-model to be migrated needs to be annotated to indicate occurrences of multi-level modelling "smells" [6]. Then, this meta-model is automatically transformed into a multi-level neutral representation that is able to accommodate some of the most prominent MLM approaches. From this representation, a number of heuristics recommend the best suitable MLM tool for the given problem, for which a serializer synthesizes the multi-level artefact. In this short paper, we present an overview of the steps in the process, and a preliminary evaluation on some meta-models developed by third parties. *Paper organization*. Section 2 introduces background on MLM and a running example. Then, Section 3 describes the rearchitecture process, and Section 4 shows a preliminary evaluation. Finally, Section 5 compares with related work, and Section 6 concludes.

2 Background and motivation

This section illustrates the main concepts of potency-based multi-level modelling, based on an example in security policies and its encoding using either two or multiple levels.

Mouelhi et al. [14] propose a meta-model to represent access control languages (like RBAC or OrBAC) and security policies described with them. An excerpt of it is shown in Fig. 1(a). Hence, the meta-model contains elements to represent both RuleTypes and Rules, and parameter types (class ElementType) and parameter instances (class Parameter). These are two occurrences of the type-object pattern, and arise due to the need to model both types and instances at the same meta-level. In this way, the conformity relation between instances and types is reified by the three associations named type.

This solution uses classes to represent both types and instances because it assumes just one instantiation level below. Instead, should we be able to use more than two meta-levels, a simpler solution like the one in Fig. 1(b) would suffice. This model has potency 2 (indicated by the "@" symbol), which means that it can be successively instantiated at the two subsequent meta-levels. Each element inside the model receives the potency of its container element, if no specified otherwise. For example, ElementType has potency 2, and so it can be successively instantiated at the two next meta-levels. In contrast, ElementType.hierarchy has potency 1, so it can only be instantiated one level below. This multi-level model specification is roughly half the size than the flat meta-model (3 vs. 6 classes, 4 vs. 7 attributes, 4 vs. 10 associations, and 3 vs. 0 potency marks).

Fig. 1(c) shows an instance of the meta-model in Fig. 1(a). It contains a small part of the definition of the RBAC language and an example of use. Hence, it defines rule type UserRole, and one instance of it named R1.

Fig. 1(d) shows the equivalent multi-level version making use of two meta-levels. The upper model (with potency 1) contains the definition of the RBAC language, while the lower model (with potency 0) defines the RBAC instance. The elements in the multi-level model (e.g., rbac) are instances of a type (e.g., PolicyType), and types w.r.t. other

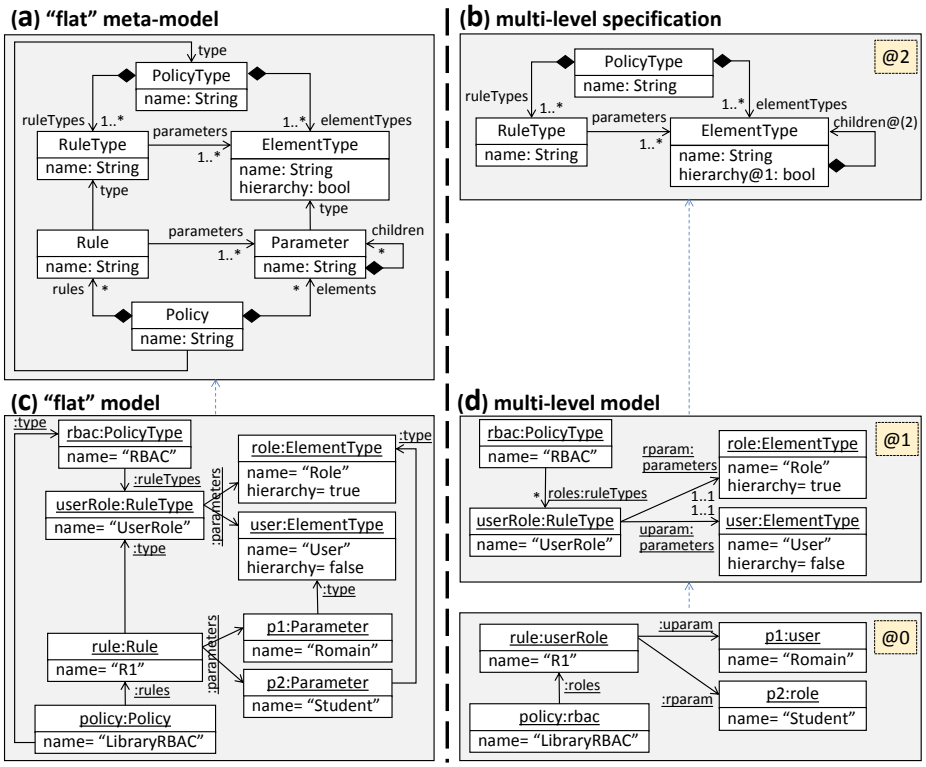

Fig. 1. Security policies: (a,c) two-level solution, (b,d) multi-level solution.

elements (e.g., policy). This way, they have both a type and an instance facet, and so they are called clabjects (merging of the words *cla*ss and o*bject*) [3]. This duality also applies to associations. This way, in the model with potency 1, associations rparam and uparam can declare cardinalities which (by default) apply to the next meta-level only. This possibility of defining cardinalities leads to a more precise model.

Altogether, in this case, the multi-level solution yields a simpler language definition (see Fig. 1(a-b)). Moreover, it permits organising models across meta-levels (see Fig. 1(c-d)), hence providing separation of concerns between language designers (e.g., the RBAC language designer within the security policies domain) and language users.

While this example is small and easy to refactor into a multi-level solution by hand, meta-models may be large, and then, their refactoring into multi-level becomes error-prone. Hence, the next section describes our approach for their automated refactoring.

3 Rearchitecting meta-models into multi-level solutions

Fig. 2 shows our process to rearchitect a meta-model into multi-level. First, the meta-model needs to be analysed to discover "smells" that indicate the convenience of migrating into a multi-level solution. These smells include the type-object pattern, and others identified in [6]. Although we currently perform this analysis by hand, it could

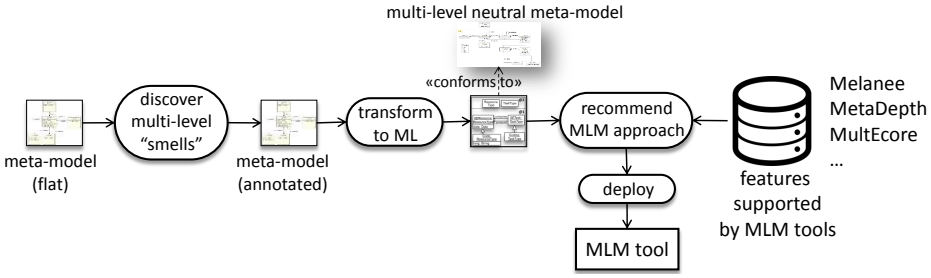

Fig. 2. Our process to rearchitect a meta-model into multi-level

be semi-automated using heuristics. The presence of smells is signalled by annotating the involved meta-model elements.

In a second step, we transform the annotated meta-model into an instance of a multi-level neutral meta-model. A recommendation system analyses this model to detect the features supported, not supported, or which can be emulated by a number of MLM tools. The result is a ranked list of candidate tools. When one of such tools is selected, the neutral model is serialized into the specific format of the tool. The overall process is extensible with new smells and MLM tools.

Next, we explain the steps of the process.

1) Discovery of multi-level smells. The first step is to annotate the occurrences of multi-level patterns in the meta-model. We take as a basis the patterns identified in [6]. These include the type-object pattern, where a class plays the role of *type*, another the role of *instance*, and a relation between them the role of *typing*. As Fig. 3 shows, the running example contains three of such occurrences. The pattern also applies to associations, in which case, their source and target classes should be in a type-object relation. Fig. 3 contains three occurrences, like association RuleType.parameters which plays the role of type for Rule.parameters.

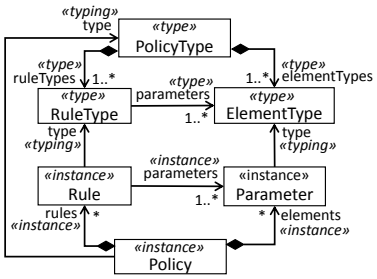

Fig. 3. Annotated meta-model

Several heuristics are possible to automatically detect occurrences of the type-object pattern, e.g., based on naming conventions (see pairs $\langle Name \rangle Type$ / $\langle Name \rangle$ in Fig. 3). However, at this stage, we have focussed in the translation of different variations of this pattern into a MLM setting, leaving pattern detection heuristics for future work.

One of such variations is the *static types* [6], where a superclass plays the role of type for a subclass, and the typing relation is reified as inheritance. In this case, refactoring into multi-level enables the dynamic creation of instances of the type class.

2) Transformation into a multi-level neutral representation. We transform the annotated meta-model into an instance of the multi-level neutral meta-model shown in Fig. 4(a). It is "neutral" as it captures generalizations of multi-level concepts found in MLM tools like Melanee, METADEPTH or MultEcore.

In this neutral meta-model, each element may have a potency governing its instantiation. In order to account for different semantics, our potency extends the classical notion [3] (explained in Section 2) with an interval [start..end]. This specifies a range

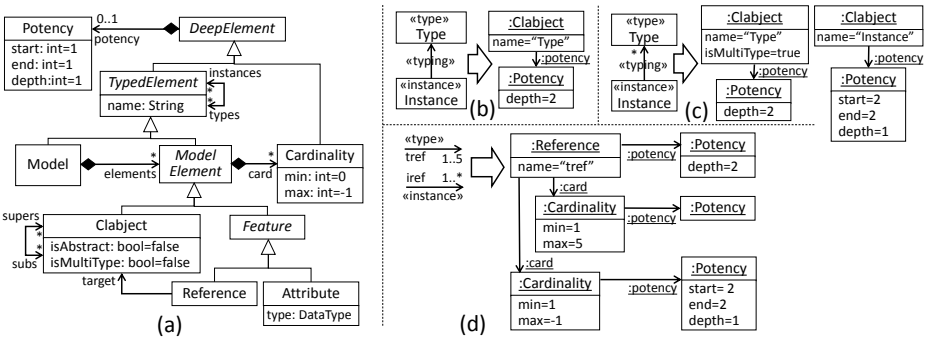

Fig. 4. (a) Multi-level neutral meta-model. Transformation of: (b) basic type-object pattern; (c) type-object with optional, multiple typing; (d) type-object of references

of meta-levels where it is possible to create direct instances of the element. The number of subsequent instantiations of these instances is governed by depth, as in classical potency. The default value for the potency interval is [1..1], and the default depth is 1, which means that the element can be instantiated in the next meta-level but these instances cannot be instantiated further. This corresponds to standard two-level modelling. An interval [2..2] means that the element can be instantiated starting two levels below, and its combination with depth 1 corresponds to the notion of *leap potency* [6]. We will explain further combinations of values in the *recommending* step.

Transforming a meta-model into our multi-level neutral representation may yield a model spanning one or several meta-levels. These are reified using the Model class. Models can hold Clabjects and Features. To account for MLM approaches supporting elements with multiple types, TypedElement declares the multi-valued reference types, and Clabject defines the flag isMultiType. Model elements can have zero or more Cardinality restrictions, constraining the number of instances that can be created at a certain level (not necessarily the next one, as in two-level modelling). The level to which the cardinality restriction applies is indicated by assigning a potency to it.

Figs. 4(b), 4(c) and 4(d) show the transformation of the meta-model annotations for some representative variants of the type-object pattern. Fig. 4(b) corresponds to the base case, where the instance class has a single class type. This is transformed into a clabject for the type, with default potency interval [1..1] and depth 2. This captures the possibility of creating types in the next level, and instances of them two levels below. The running example has three occurrences of this base case. Fig. 4(c) tackles the situation where the instance class can have multiple, optional typing. Multiple typing is handled as the base case, but the isMultiType attribute of the produced clabject is set to true. Optional typing requires producing another clabject for the instance, with potency interval [2..2] and depth 1. This enables the leap instantiation of instances two levels below, thus emulating instances with no type. We also support other variations in the direction and cardinality of the typing relation, as well as its realization as an attribute identifier. The *static types* variant aforementioned, where the type and instance classes are related through inheritance, is also supported. This case leads to one model for the type class and another for the instance class, which are related through instantiation.

Table 1. Support of MLM concepts by tools: native support (+), emulated (∼), unsupported (−).

dimension	multi-level feature	start	end	depth	Melanee	MetaDepth	MultEcore
potency in clabjects and references	standard potency	1	1	n (n≥1)	+	+	∼ (n=∞)
	leap potency	n (n>1)	n	1	∼ (n=1)	+	∼ (n=∞)
	replicability	n (n≥1)	m (m>n)	1	− (n=1,m=n)	− (m=n)	+
	deep leap potency	n (n>1)	n	o (o>1)	− (n=1,o=1)	− (o=1)	∼ (o=∞)
	deep replicability	n (n≥1)	m (m>n)	o (o>1)	− (n=1,m=n,o=1)	− (m=n,o=1)	∼ (o=∞)
potency in attributes	attribute durability	1	n (n≥1)	1	+	+	∼ (n=∞)
	attribute mutability	n (n≥1)	m (m>n)	1	+	∼ (m=n)	− (always mutable)
instantiation	shallow ref. cardinality	1	1	1	+	+	+
	deep ref. cardinality	n (n>1)	m (m>n)	o (o>1)	−	∼ (OCL)	−
	multiple typing	-	-	-	−	+ (a-posteriori)	+ (supplementary)
	abstract types	-	-	-	∼ (potency=0)	+	+

The transformation also handles the type-object pattern applied to references, as Fig. 4(d) shows. This creates one Reference with potency depth 2, owned by the clabject declaring the reference. Moreover, the cardinality bounds get transformed into two Cardinality objects. The first one captures the cardinality of the type reference and has potency 1, being enforced one level below. The second one comes from the instance reference and has leap potency 2, being applied two levels below.

As regards to the meta-model elements with no annotation, they are transformed into clabjects or features with default attribute values.

3) Recommendation of MLM approach. After transforming the meta-model into a multi-level neutral model, we analyse it to identify the multi-level features required for the problem at hand, and which tools provide support for them. For this purpose, we have built a recommender that recognizes the required multi-level features by detecting certain patterns on the configuration of element potencies, and then counts how many of such features are either natively supported, can be emulated, or are unsupported in each tool. The recommender yields a ranking according to the number of features natively supported, and in case of tie, by the number of features that can be emulated. It also reports the unsupported required features.

Table 1 contains a summary of the patterns sought by the recommender (columns multi-level feature, start, end and depth). The last three columns show whether these features are supported, unsupported or can be emulated by three representative tools: Melanee, METADEPTH and MultEcore.

All tools support or can emulate *standard potency*, while METADEPTH natively supports *leap potency*, which is necessary for models like the one in Fig. 4(c). Our multi-level neutral meta-model supports other variations of potency like *replicability*, where an element can be instantiated in a range of levels with depth 1, *deep replicability*, where in addition the created instances can be further instantiated, or *deep leap potency*, where instantiation starts after a level gap and can be iterated. Interestingly, none of the three tools fully support the last two options.

Regarding attributes, *durability* indicates how many levels below an attribute is instantiated, and *mutability* the range of levels where an attribute can be modified. Melanee natively supports both, while they can be emulated with the other two tools.

As for reference cardinalities, none of the tools permit their specification for levels beyond the next one, although METADEPTH can emulate this via OCL. All tools support abstract types (Melanee emulates them via clabjects with potency 0), while multiple types are possible in METADEPTH and MultEcore.

Table 2. Results of applying our rearchitecting process to some of models analysed in [6].

meta-model	flat meta-model			multi-level version				type-object occur.		tool ranking					
	classes	refs.	attrs.	classes	refs.	attrs.	reduc.	clabject	assoc.	Melanee		MetaDepth		MultEcore	
Security Policies	7	12	7	4	5	4	50%	3	4	1	✓	2	✓	3	×
Agate	69	123	81	64	118	81	4%	6	2	2	✓	1	✓	3	✓
CloudML	21	28	26	15	17	26	23%	6	7	2	✓	1	✓	3	✓
CloudML-2.0	33	50	44	21	40	44	18%	12	5	2	✓	1	✓	3	✓
HAL	42	16	72	41	15	72	2%	1	0	2	✓	1	✓	3	✓

4 Experiments

We have developed a prototype tool with a recommender for Melanee, METADEPTH and MultEcore. We have used the tool to perform an initial evaluation of our process by rearchitecting the running example and four third-party meta-models which contain occurrences of the type-object pattern, as identified in [6]. Table 2 shows the size metrics before and after the rearchitecture, as well as the reduction percentage in the number of elements (clabjects, references and attributes) required to express the same information. The reduction ranges between 2% (HAL) and 50% (running example). The small reduction size for HAL and Agate is because these are the biggest meta-models and have few type-object occurrences. The reduction gain in the rest of cases is considerable.

The table also includes the number of type-object occurrences in each meta-model, and the ranking of recommended tools based on their support of the features in Table 1. This score adds 2 points for each required multi-level feature that the tool supports natively, 1 point if the feature can be emulated, and -1 if it is unsupported. We have marked with a cross the cases in which a tool does not support some required feature.

The recommended tool for the running example is Melanee, as it supports mutability for the three attributes name, and it can emulate leap potency (required for reference children). METADEPTH appears in second place because it has native support for leap potency, but mutability needs to be emulated. MultEcore is in third place as it does not support attribute mutability as required in this case.

More details on the evaluation can be found at `http://miso.es/MLeval`.

5 Related work

We are not aware of any effort for the automatic rearrangement of meta-models into multiple levels. There are just some works on manual rearchitecture [6, 9], or introducing concrete multi-level elements [10]. In [6], we identified patterns where using MLM may have benefits. In [9], a standard in the oil & gas industry is recasted into multiple levels. In [10], some evolution operators allow applying the powertype pattern in models, which may imply reorganizing elements across levels. Hence, to our knowledge, ours is the first proposal towards an extensible, automated process to migrate to multi-level (though currently only the refactoring into multi-level is fully automated, but the discovery of multi-level smells is manual).

Regarding our level-neutral meta-model, the main contribution with respect to other meta-models (e.g., in METADEPTH [5] or Melanee [2]) is the generalization of potency, the possibility to specify cardinalities for levels beyond the immediate lower one, and the capability to indicate multiple typings.

6 Conclusions and future work

We have presented an approach for the automated rearchitecture of meta-models into multi-level specifications. The approach is based on the identification of multi-level smells, and their translation into a neutral multi-level model that can be analysed to recommend the most suitable MLM tool to transform to. We have created prototype tool support, and performed a preliminary evaluation obtaining promising results.

In the future, we plan to define annotations for other multi-level smells, and heuristics to induce them. We also plan to refine the recommendation process (e.g., to consider the cost of emulating non-native features), to migrate models together with their meta-models, and to perform a large scale evaluation.

Acknowledgements. Work supported by the Spanish MINECO (TIN2014-52129-R) and the R&D programme of the Madrid Region (S2013/ICE-3006).

References

1. C. Atkinson, R. Gerbig, and M. Fritzsche. A multi-level approach to modeling language extension in the enterprise systems domain. *Inf. Syst.*, 54:289–307, 2015.
2. C. Atkinson, B. Kennel, and B. Goß. The level-agnostic modeling language. In *SLE*, volume 6563 of *LNCS*, pages 266–275. Springer, 2010.
3. C. Atkinson and T. Kühne. Reducing accidental complexity in domain models. *SoSyM*, 7(3):345–359, 2008.
4. T. Clark, P. Sammut, and J. S. Willans. Super-languages: Developing languages and applications with XMF (second edition). *CoRR*, abs/1506.03363, 2015.
5. J. de Lara and E. Guerra. Deep meta-modelling with METADEPTH. In *TOOLS*, volume 6141 of *LNCS*, pages 1–20. Springer, 2010. See also `http://metaDepth.org`.
6. J. de Lara, E. Guerra, and J. Sánchez Cuadrado. When and how to use multi-level modelling. *ACM Transactions on Software Engineering and Methodology*, 24(2):12, 2014.
7. U. Frank. Multilevel modeling - toward a new paradigm of conceptual modeling and information systems design. *Business & Information Systems Engineering*, 6(6):319–337, 2014.
8. R. Gerbig, C. Atkinson, J. de Lara, and E. Guerra. A feature-based comparison of melanee and metadepth. In *Proc. MULTI@MODELS*, volume 1722 of *CEUR*, pages 25–34, 2016.
9. M. Igamberdiev, G. Grossmann, M. Selway, and M. Stumptner. An integrated multi-level modeling approach for industrial-scale data interoperability. *SoSyM*, to appear:1–26, 2016.
10. M. Jahn, B. Roth, and S. Jablonski. Remodeling to powertype pattern. In *Proc. PATTERNS*, pages 59–65, 2013.
11. M. A. Jeusfeld and B. Neumayr. DeepTelos: Multi-level modeling with most general instances. In *Proc. ER*, volume 9974 of *LNCS*, pages 198–211, 2016.
12. Y. Lamo, X. Wang, F. Mantz, Ø. Bech, A. Sandven, and A. Rutle. Dpf workbench: a multi-level language workbench for mde. *Proc. Estonian Academy of Sciences*, 62(1):3–15, 2013.
13. F. Macías, A. Rutle, and V. Stolz. MultEcore: Combining the best of fixed-level and multi-level metamodelling. In *MULTI@MODELS*, volume 1722 of *CEUR*, pages 66–75, 2016.
14. T. Mouelhi, F. Fleurey, and B. Baudry. A generic metamodel for security policies mutation. In *Proc. ICST*, pages 278–286. IEEE Computer Society, 2008.
15. B. Neumayr, C. G. Schuetz, M. A. Jeusfeld, and M. Schrefl. Dual deep modeling: multi-level modeling with dual potencies and its formalization in f-logic. *SoSyM*, to appear:1–36, 2016.
16. M. Selway, M. Stumptner, W. Mayer, A. Jordan, G. Grossmann, and M. Schrefl. A conceptual framework for large-scale ecosystem interoperability and industrial product lifecycles. *Data & Knowledge Engineering*, in press:1 – 27, 2017.