CEUR-WS.org/Vol-2245/multi_paper_1.pdf

A tool for the convergence of multilevel modelling approaches

Fernando Macías[1,2], Adrian Rutle[1], and Volker Stolz[1,2]

[1] Western Norway University of Applied Sciences
{first.last}@hvl.no
[2] University of Oslo

Abstract Multilevel Modelling is a powerful paradigm that can improve the way we create and use models. The community and approaches related to Multilevel Modelling have been constantly growing, and the need to agree on some basic concepts, semantics and vocabulary has become paramount. In this paper, we present a tool that can provide empirical data and practical instrumentation to aid in the discussion of the foundational concepts of the paradigm. We outline the structure, modules and applications of the tool in detail, and explain how they can be employed for different tasks that can benefit the Multilevel Modelling community. To illustrate these benefits, we also describe several experiments carried out with the tool and their positive results.

1 Introduction

The field of Multilevel Modelling (MLM) has crystallised in recent years as a relevant field inside the Model-Driven Software Engineering (MDSE) community. Concepts like multiple levels of abstraction, deep instantiation and inter-model constraints have been proven as valid solutions to scenarios where traditional fixed-level modelling lacks expressiveness [10]. Consequently, the number of proposals and tools in the field of MLM has grown considerably. However, some of these concepts, as well as their semantics, remain to be agreed upon by the community. This lack of consensus is noticed in recent examples like the Bicycle Challenge [7], aimed at showcasing and comparing different approaches to MLM; the challenges and need of discussion described in [4]; a comparison between two prominent MLM tools [11]; and even the description of the MULTI workshop series on its website[3].

Our aim is to contribute to the convergence of MLM concepts and their semantics, so that a common base of knowledge is created and the discussions about MLM can progress towards more advanced topics. We build upon the work presented in [12], where a prototype of a tool for rearchitecting Ecore metamodels into multilevel hierarchies was introduced. For that rearchitecting process, we identified certain patterns in the metamodels (so-called "smells" [10]) and transformed them to encode the same information into more concise multilevel hierarchies. We present here new additions to the original proposal and highlight the advantages which they provide and can be relevant for the MLM community, such as the interoperability and comparison of MLM tools.

[3] https://www.wi-inf.uni-duisburg-essen.de/MULTI2018/#about

The structure and main contributions of this paper are: (1) a detailed presentation of an intermediate, tool-agnostic metamodel that can be used to represent multilevel hierarchies (Section 2); (2) the addition of importing capabilities so that models defined in different tools can be transformed into instances of the tool-agnostic metamodel; this is to facilitate their comparison, interoperability, evaluation and improvement, as well guide the choice of suitable tools for a specific scenario (Section 3); and (3) the description and results of an experiment using these import and export capabilities, drawing from our own experiences and illustrating the usefulness of the process (Section 4). Finally in Section 5 we present some related works, summarise the main results of this paper and draw future lines of research that can stem from our contributions.

2 Tool-agnostic MLM metamodel

The tool-agnostic Ecore metamodel depicted in Figure 1 contains the concepts that are used to represent a multilevel, tree-shaped hierarchy of models, which can also be used to represent a single stack with only one model per level. This metamodel aims to include all features from three MLM tools and be open for the inclusion of new ones. In its current shape, it has been constructed by some of the proponents of the included tools based on their knowledge and experience.

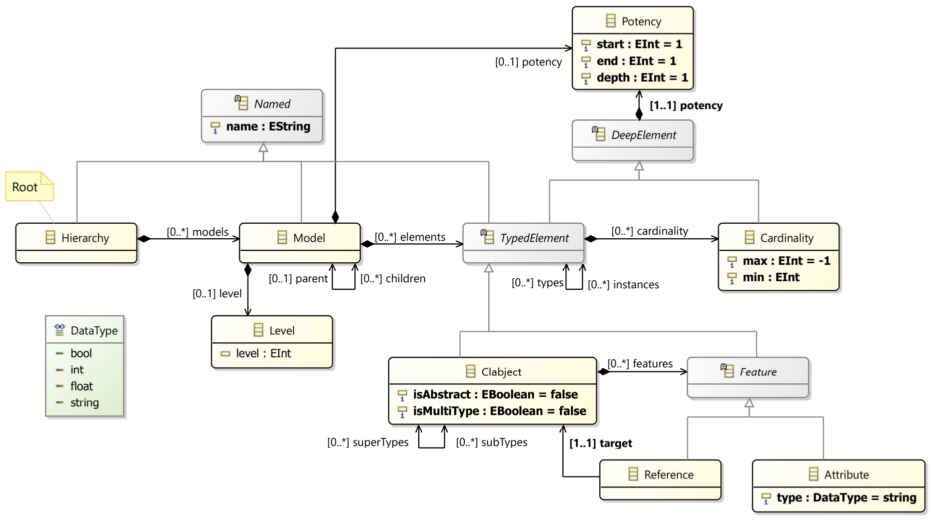


Figure 1: Metamodel to represent tool-agnostic multilevel hierarchies

The tool-agnostic metamodel has a root class, called Hierarchy that acts as the root (required in EMF) and contains a list of models. This list contains Model elements, that can have at most one metamodel (reference parent) and any number of instance models (reference children). These two references ensure the aforementioned tree shape of hierarchies. A model can have a Level, represented as a class containing an integer attribute whose default value is zero. This level information is required by some MLM tools, and the numbering convention may differ from tool to tool. A Model may also have Potency, which in some cases is used as the default potency of the elements it contains, as explained below. Both hierarchies and models must have a name that can be used to identify them.

All elements that can be contained inside a Model inherit from the abstract class TypedElement. By doing so, we ensure that all the concepts that we present in the following will have a name (since TypedElement extends Named) and a potency (by extending DeepElement). Typed elements can have any number of types—allowing for multiple typing, which is supported by some MLM tools—and therefore any number of instances. Furthermore, the required Potency for all elements is used to control where, and how many times, an element can be instantiated. The design decision to have it as a separate class is to favour composition over inheritance and keep the metamodel (and related code) flexible in case of future modifications, and the same applies to LEVEL. In order to cover all potency possibilities provided by the three tools considered so far—Melanee [3], MetaDepth [9] and MultEcore [13]—the potency is specified by three values: start, to indicate the minimum number of levels below where the element can be instantiated (minimum value is 1); end, to indicate the maximum number of levels below where the element can be instantiated (this value must be bigger or equal than start); and depth, to indicate the number of times an instance of the element can be instantiated at the levels below. That is, the depth of an instance will always be one less than the depth of its type. For example, if an element A at level 1 has potency 2,3,2, then A can be instantiated in levels 3 and 4 (the instances will have depth 1), and instances of those instances can also be created, but not instantiated any further (since their depth will be 0).

A Model can contain Clabjects, representing the class-object duality of the element, as initially defined in [2]. Clabjects can be abstract (meaning they cannot be instantiated) and have multiple types. This is indicated by the two boolean attributes it contains, which are set to false by default. The possibility of a clabject acting as superclass of another (inheritance relation) is indicated by the superTypes reference and its opposite, subTypes. As specified by the cardinalities of these references, multiple inheritance is allowed. Clabjects can contain two kinds of Features, namely Reference and Attribute. The former relates its containing clabject (source) with another clabject, indicated by target, whereas the latter specifies an attribute of the clabject, where a type must be specified as one of the given data types in the DataType enumeration. These data types are limited for now to the most common four: boolean, integer, float and string (the default one). Both Clabject, Reference and Attribute can have any number of cardinalities (optional for now), which control the minimum and maximum number of instances that can be created of the corresponding element. The boundaries are given by the cardinality's two integer attributes, min (with default value 0) and max (default value -1, i.e. unbounded). These cardinalities must also define a potency (by inheriting from DeepElement) which specifies in which levels each cardinality must hold.

The purpose of this metamodel is to act as an internal representation for the rearchitecting process described in Section 3. Besides, it can be used as a reference for key features that are expected from multilevel-aware modelling tools, and as a basis for discussion about those features: which ones should be included and their semantics.

3 MLM Rearchitecter tool

In this section, we describe our MLM rearchitecting tool, with special focus on the parts relevant to this paper and the new additions. The original version of this tool was presented in [12], as an implementation of some of the ideas from [10] for detecting "smells" in two-level metamodels and refactoring them into MLM patterns.

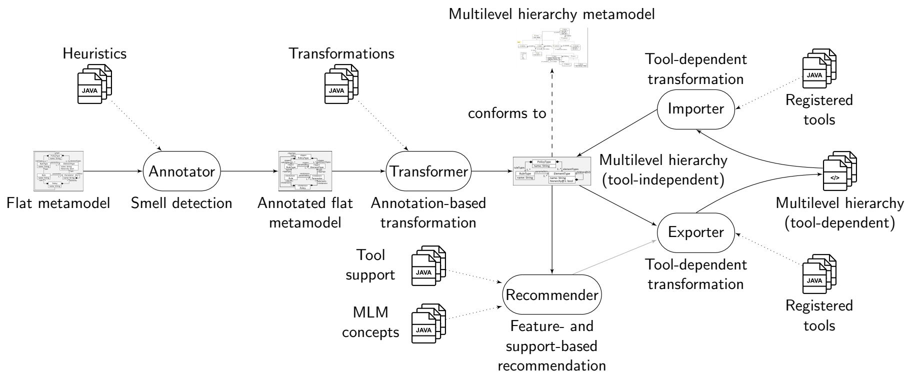

Multilevel hierarchy metamodel
Heuristics
Transformations
Tool-dependent transformation
conforms to
Importer
Registered tools
Annotator
Transformer
Multilevel hierarchy (tool-independent)
Flat metamodel
Smell detection
Annotated flat metamodel
Annotation-based transformation
Multilevel hierarchy (tool-dependent)
Exporter
Tool support
Tool-dependent transformation
Recommender
MLM concepts
Feature- and support-based recommendation
Registered tools

Figure 2: Rearchitecting process overview, updated from [12]

The rearchitecting process has a pipeline-like structure, as shown in Figure 2. The process can be executed automatically in a single run or in a step-by-step basis, allowing to fix errors, make adjustments or complete the results of any step throughout the process. Each of the modules, as well as the complete tool which encompass all of them, the Rearchitecter itself, are available in the form of Java JAR files, which can be downloaded from the project's website [16]. The interaction with the different modules is made by a console interface, and allows for different execution modes—single-file and batch—and additional output like debugging information. The tool also outputs statistical data about some of the processes, as explained below. The modules that compose the whole rearchitecting process follow a pipeline-like structure, and are described in the following.

- **Annotator** Looks for bad design "smells" in Ecore metamodels, and uses EAnnotations to mark them, together with a confidence level. Different techniques like structural patterns, string similarity for names and semantic proximity are used in of the already implemented heuristics. The full description of smells can be found at [10]. This module contains the interface *Heuristic* which can be implemented by new Java classes in order to find new smells, as indicated in Figure 2. As a result of this step, we obtain the same Ecore metamodel from the input, enriched with annotations in the elements that conform a smell and a confidence level, normalised between 0 and 10.
- **Transformer** Takes an annotated Ecore metamodel as input. Based on the information carried by these annotations—which are obtained in the previous step and can be extended or corrected manually—different transformations can be applied, in

order to turn the elements in the metamodel (e.g. an EClass) into instances of elements from our tool-agnostic metamodel (e.g. an instance of Clabject). The current state of this module already implements transformations for the currently supported annotations and, as in the previous module, can be easily extended by implementing the existing *Transformation* Java interface. To ease the task of implementing new transformations, this module contains a registry of the already transformed elements, to ensure traceability from input to output and the combination of different transformations. For example, this registry is necessary when transforming an EReference, in order to locate the clabjects that correspond to its source and target EClasses. Furthermore, this module contains default *copy* transformations, that are applied if no other transformation is suitable for a given element. The result of this process is a model representing a multilevel hierarchy that optimizes the original metamodel, and which is an instance of the metamodel presented in Section 2. It is also important to notice that the process can be adjusted to only treat annotations with a confidence level above a given threshold: some early experiments, briefly explained below, have been used to compare the automatically annotated Ecore metamodels against manually annotated ones; the experiments have shown that a confidence threshold of 7 gives the best value for precision and recall, and that is the value currently configured for this step.

– **Recommender** This module takes an instance model of the tool-agnostic metamodel, which can be generated by the previous step or by importing (explained below), and ranks the registered multilevel tools based on their suitability for modelling the hierarchy represented by the model. This ranking is calculated by counting the number of multilevel features that appear in the model, and whether the MLM tools can support them (2 points), emulate them (1 point) or do not support them (-1 point and warning). The ranking is both written as console output and as a CSV file; some of these results can be found online [16]. This information provides an empirical way of comparing MLM tools, and their suitability for different scenarios. As with the previous modules, Figure 2 also shows how the Recommender can be easily extended: contributors can both register new tools by listing their degree of support for all MLM features already defined (by implementing the *Tool* interface), as well as defining new MLM concepts and implementing how to find them in the model (implementing the *Concept* interface).

– **Exporter** Since the rearchitecting tool itself does not provide any custom editors for MLM, this module allows to transform the tool-agnostic representation into a tool-dependent one. The nature of the generated hierarchy depends completely on the selected tool, so it is the responsibility of the contributor to respect the format used for the target representation. For example, MetaDepth requires text files with a JSON-like syntax and the *mdepth* file extension. In order to add exporting options for additional tools, this module provides the *Format* interface. Any class implementing the interface will be automatically included in the tool's catalogue of registered exporters and be accessible for exportation. Apart from the existing MetaDepth exporter, the new version of the tool has been extended with another implementation for the tool MultEcore.

The pipeline conformed by the four aforementioned modules can also be used as a single black box module, called **Rearchitecter**. This module takes any Ecore metamodel as an input and outputs the representation of the equivalent multilevel hierarchy in any number of selected formats. If executed this way, the ranking information provided by the Recommender can be used to automatically select the best scoring tool as target for exportation, in addition to the default manual selection. This dependence is also depicted in Figure 2 with a grey arrow. As with the separate modules, the Rearchitecter also outputs optional debugging information and the CSV file generated in the recommendation step. Additionally, the Rearchitecter generates a second CSV file with information about the reduction achieved by the process, in terms of classes, references and attributes. This data is used to empirically assess the validity of the approach.

In parallel to the rearchitecting process, we also implemented an experiment for calculating the precision and recall of the heuristic-driven, automatic annotations against manually-annotated metamodels. We run the experiment for all potential threshold values, ranging from 0 (best recall, worst precision) to 10 (worst recall, best precision), and found the optimal value to be 7. The complete results for 35 Ecore metamodels from the AtlanMod Metamodel Zoos [6] can be found in an online table where the 11 CSV files resulting from the experiment have been imported and synthesized [16]. As mentioned before, we used this information to evaluate the correctness of the process and adjust the tool's confidence threshold for executing a specific MLM transformation in the Transformer.

Finally, in this work we present a new module for the rearchitecting tool, which provides new ways to interact with the tool-agnostic metamodel presented in Section 2.

– **Importer** This new module was created in order to increase the usefulness of the rearchitecting tool for the MLM community, by allowing the exchange of representation from tool-dependent to tool-independent. As with the previous modules, extending the module with new MLM tool compatibility is straightforward. By implementing the *Format* interface it provides, the module can automatically register it, and offer it as a possibility for importing from that MLM tool representation. Similar to the Exporter, it is the contributor's responsibility to ensure the proper exchange of representation, both in parsing the tool-dependent representation and generating a valid tool-agnostic model instance. Also similar is the fact that we did not use any model transformation or model weaving techniques (such as **??**) since it cannot be guaranteed that the tools will have a metamodel for their representation format, or that it will be compatible with EMF technologies. A proof of concept implementation for the tool MultEcore is included in the current version of the module. A small experiment presented in Section 4 shows how this new module allows for an exchange of representation between MultEcore and MetaDepth and a "round-trip" for MultEcore.

4 Experiments and results

In this section, we present some initial experiments and experiences with the rearchitecting tool, with special focus on the newly added Importer module and its capabilities.

Table 1: Repetition of the experiment from [12], including the results after improving MultEcore: best scores in bold and tools with unsupported features in red

Hierarchy	**Size (multilevel)**				**Melanee**	**MetaDepth**	**MultEcore (old)**	**MultEcore (new)**
	Models	**Classes**	**Refs.**	**Attrs.**				
Security Policies	1	4	5	4	31	29	10	**32**
Agate	1	64	118	81	398	**518**	279	**518**
CloudML	1	15	17	26	111	**120**	64	**120**
CloudML-2.0	1	21	40	44	188	**214**	112	**214**
HAL	1	41	15	72	256	**284**	156	**284**

Improving score in the Recommender In earlier experiments with the Rearchitecter, the tools Melanee, MetaDepth and MultEcore were compared. MultEcore was the most recent one, and hence its score was always the lowest. In Table 1, columns 6, 7 and 8 show the results from those first experiments (repeated from[12]). Thanks to the possibility of comparing empirically with third-party examples, we identified the most necessary improvements that needed to be made to MultEcore. These were, mainly, the necessity to introduce the depth value into potencies—hence becoming three-valued—, the way to handle attributes and the semantics of inheritance. After these improvements, the support of several multilevel features in MultEcore changed, and a repetition of the experiment showed that the tool is now as capable as the other two, and even better in one case. The improved scores can be found in the last column of Table 1. We believe that other tools can also benefit from this empirical evaluation, especially in the case of new proposals for multilevel-aware frameworks.

Evaluating MultEcore's examples One of the main contributions of the Importer module is allowing to represent examples made specifically for a multilevel modelling tool—normally aimed at showcasing that tool's strengths—in the tool-agnostic format. This way, the creators of the tool can empirically validate the claimed strengths of their approach. To illustrate this, we used several example hierarchies that were previously developed for different projects, available at the MultEcore project repositories[4]. Additionally, this experiment helped test the new Importer module with its implementation for MultEcore. The process followed in this experiment consisted firstly in using the Importer to represent the multilevel hierarchies in our examples as instances of the tool-agnostic metamodel. Once this was done, we used the Recommender to create a similar table to the one previously showed. For this second experiment, the results are depicted in Table 2. As expected, MultEcore obtains the highest scores in the most complex examples, and ties with Melanee and MetaDepth in the simplest ones, validating the strengths of MultEcore in certain scenarios.

Import-Export for the same tool With the new addition of the Importer, it is now possible to transform back and forth among different representations, or even within the same representation. To check that no information is lost, we imported a small hierarchy, and obtained its tool-agnostic representation. The example is an excerpt of the solution

[4] `https://bitbucket.org/account/user/phd-fernando/projects/MUL`

Table 2: Results for each tool after importing and recommendation: best scores in bold and tools with unsupported features in red

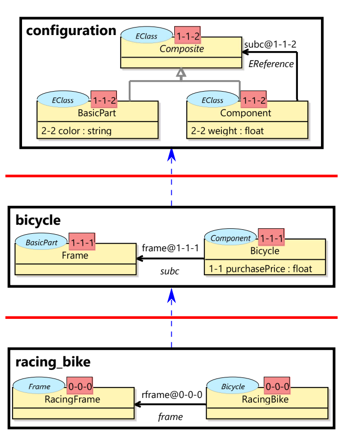
Hierarchy	**Size (Multilevel)**				**Melanee**	**MetaDepth**	**MultEcore**
	Models	**Classes**	**References**	**Attributes**			
bicycle	6	32	17	17	125	119	**140**
datatypes	3	21	27	0	**96**	**96**	**96**
ltl	5	59	35	4	**196**	**196**	**196**
petrinets	4	20	21	3	85	82	**94**
pls	5	32	25	0	105	105	**114**
robolang	7	69	78	1	215	214	**296**

presented in [14] to the first edition of the Bicycle Challenge from the MULTI 2017 workshop (see the new edition at [7]). The initial version of the hierarchy in MultEcore is depicted in Figure 3a, which contains three models. After running the importer, we obtained the tool-agnostic representation, depicted in Figure 3b. Re-exporting into MultEcore, using the Exporter, yields the same three initial models from Figure 3a, hence demonstrating—although not formally proving—that no information is lost or corrupted for small hierarchies when importing and exporting, and that the tool-agnostic metamodel is capable of representing multilevel hierarchies.

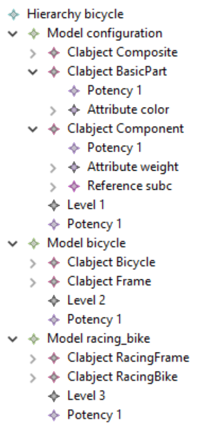
Import-Export for different tools In a similar fashion to the previous experiment, we tested the capabilities of the tool as a future "exchange format" to be used among multilevel modelling tools. To this end, we used the imported version of the example hierarchy from the previous experiment (Figure 3b), and exported to a different MLM tool than the original one; i.e. to MetaDepth instead of MultEcore. By doing this, we manually compared the initial version of the hierarchy (Figure 3a) against the exported version (Figure 3c), and verified that the same models, classes, references and attributes were created; and that their properties (being abstract, potencies, cardinalities, etc.) were preserved. It is important to note that the initial version of the hierarchy was adapted to ensure that it would be completely compatible with MetaDepth. The Recommender module was helpful for this task, since it informs whether a tool supports all features of a given hierarchy or not, as previously explained.

5 Conclusions

To the best of our knowledge, this is the first effort dedicated towards a tool-supported common representation of multilevel hierarchies, including import/export capabilities that can evolve into a fully fledged exchange format for MLM tools. However, there are other works worth mentioning with similar goals. One of them is the conceptual classification of architectures described in [5], where fixed-level and multilevel conceptual frameworks are presented and exemplified. In [11] the creators of of Melanee and MetaDepth made a feature-based comparison of both tools, and mention transformations for exchanging between them as a work in progress. Finally, in [15] the authors compare the realisation of potency and the conceptual basis of several MLM tools. All these works, as well as the one presented here, focused only on the functional aspects of the tools. We



(a) MultEcore



- Hierarchy bicycle
  - Model configuration
    - Clabject Composite
    - Clabject BasicPart
      - Potency 1
      - Attribute color
    - Clabject Component
      - Potency 1
      - Attribute weight
      - Reference subc
    - Level 1
    - Potency 1
  - Model bicycle
    - Clabject Bicycle
    - Clabject Frame
    - Level 2
    - Potency 1
  - Model racing_bike
    - Clabject RacingFrame
    - Clabject RacingBike
    - Level 3
    - Potency 1

(b) Tool-agnostic

```
ext Model configuration@2{
  ext abstract Node Composite@2{}
  ext Node BasicPart@2 : Composite
    {color@2:String[0..1];}
  ext Node Component@2 : Composite
    {subc@2:Composite[*];
     weight@2:double[0..1];}
}
ext configuration bicycle{
  ext BasicPart Frame{}
  ext Component Bicycle
    {purchasePrice@1:double[0..1];
     frame@1:Frame {subc };}
}
ext bicycle racing_bike{
  ext Frame RacingFrame{}
  ext Bicycle RacingBike
    {frame=RacingFrame;}
}
```

(c) MetaDepth

Figure 3: Representation exchange from MultEcore, back to MultEcore and to MetaDepth

believe that a comparison of non-functional features like usability, user-friendliness and performance might as well be useful for the MLM community in the future. Moreover, an analysis of the suitability of MLM tools for their pragmatic value and suitability for specific domains could complement the results presented in this paper.

The main contributions of this paper are aimed at fostering discussion and collaboration towards the convergence of MLM principles and semantics, a necessity already identified in previous works [4]. For this purpose, we have proposed a metamodel for the representation of multilevel hierarchies and we invite collaborations and contributions to the tool and the metamodel, which we consider to be still in an early version. Future versions should address missing concepts like instantiated attributes and constraints like non-cyclic typing. Also, we have improved the tool built around this metamodel, so that MLM tools can be compared by directly importing technology dependent MLM hierarchies into the tool-agnostic representation. In addition, the experiments we have run on several examples have shown that the evaluation helps in finding out which features a specific MLM tool should implement in order to get a higher score; hence providing a roadmap for further improvements. Based on the authors' own experience, we showed how MultEcore was improved by adding explicit support for depth, better attribute handling and inheritance, thanks to the results of these experiments.

We have also included import and export capabilities to the Rearchitecter for our own tool MultEcore as a proof of concept (in addition to the existing MetaDepth exporter implementation). The tool is nevertheless designed to be fully extensible by providing interfaces in every step of the process, making straightforward the addition of, for example, new tool importers, exporters and recommenders. This can be seen as a call for future community contributions to adapt and implement the scoring mechanisms for the recommender and the import/export capabilities for other MLM tools. The experiments we have run on several examples have shown that importing from MultEcore and exporting back to it gives an equivalent model, demonstrating the preservation of information in both processes.

We believe that another future line of work can be the formal definition of the semantics for the tool-agnostic metamodel, so that it can serve as a reference implementation, a standard and an exchange format for MLM-capable tools, in the line of previous approaches for model exchange through common representations (see, e.g. [8]). Thanks to the mapping between tools that the Rearchitecter provides, it may even be possible to unify existing tool-independent formalisations, like the ones presented in [17], [15] and [1], and adapt them to the tool-agnostic representation.

References

1. J. P. A. Almeida, C. M. Fonseca, and V. A. Carvalho. A comprehensive formal theory for multi-level conceptual modeling. In *International Conference on Conceptual Modeling*, pages 280–294. Springer, 2017.
2. C. Atkinson. Meta-modelling for distributed object environments. In *Enterprise Distributed Object Computing Workshop (EDOC'97)*, pages 90–101. IEEE, 1997.
3. C. Atkinson, R. Gerbig, and M. Fritzsche. A multi-level approach to modeling language extension in the enterprise systems domain. *Information Systems*, 54:289–307, 2015.
4. C. Atkinson, R. Gerbig, and T. Kühne. Comparing multi-level modeling approaches. In *MULTI 2014*, pages 53–61, 2014.
5. C. Atkinson and T. Kühne. Concepts for comparing modeling tool architectures. In *International Conference on Model Driven Engineering Languages and Systems*, volume 3713 of *LNCS*, pages 398–413. Springer, 2005.
6. AtlanMod Metamodel Zoos. *Web site*. `http://web.emn.fr/x-info/atlanmod/index.php?title=ZooFederation`.
7. Bicycle Challenge description. *Web site*. `https://www.wi-inf.uni-duisburg-essen.de/MULTI2018/#challenge`.
8. X. Blanc, M.-P. Gervais, and P. Sriplakich. Model bus: Towards the interoperability of modelling tools. In *Model driven architecture*, pages 17–32. Springer, 2005.
9. J. de Lara and E. Guerra. Deep meta-modelling with MetaDepth. In *Objects, Models, Components, Patterns*, volume 6141 of *LNCS*, pages 1–20. Springer, July 2010.
10. J. de Lara, E. Guerra, and J. Sánchez Cuadrado. When and how to use multilevel modelling. *ACM Trans. Softw. Eng. Methodol.*, 24(2):12:1–12:46, Dec. 2014.
11. R. Gerbig, C. Atkinson, J. de Lara, and E. Guerra. A feature-based comparison of melanee and metadepth. In C. Atkinson, G. Grossmann, and T. Clark, editors, *Proc. 3rd Intl. Workshop on Multi-Level Modelling*, volume 1722 of *CEUR Workshop Proceedings*, pages 25–34, 2016.
12. F. Macías, E. Guerra, and J. de Lara. Towards rearchitecting meta-models into multi-level models. In *International Conference on Conceptual Modeling*, pages 59–68. Springer, 2017.
13. F. Macías, A. Rutle, and V. Stolz. MultEcore: Combining the best of fixed-level and multilevel metamodelling. In *3rd International Workshop on Multi-Level Modelling (MULTI2016)*, volume 1722 of *CEUR Workshop Proceedings*, 2016.
14. F. Macías, A. Rutle, and V. Stolz. Multilevel Modelling with MultEcore: A Contribution to the MULTI 2017 Challenge. In *4th International Workshop on Multi-Level Modelling (MULTI2017)*, volume 2019 of *CEUR Workshop Proceedings*, 2017.
15. F. Macías, A. Rutle, V. Stolz, R. Rodriguez-Echeverria, and U. Wolter. An approach to flexible multilevel modelling. *Enterprise Modelling and Information Systems Architectures*, 13:10:1–10:35, 2018.
16. MLM Rearchitecting Project. *Web site*. `http://ict.hvl.no/mlm-rearchitect-results/`.
17. A. Rossini, J. de Lara, E. Guerra, A. Rutle, and U. Wolter. A formalisation of deep metamodelling. *Formal Aspects of Computing*, 26(6):1115–1152, 2014.