Resilience in Sirius Editors: Understanding the Impact of Meta-Model Changes

Juri Di Rocco[1], Davide Di Ruscio[1], Hrishikesh Narayanankutty[1,2], and Alfonso Pierantonio[1]

[1] Università degli Studi di L'Aquila
I-67100 L'Aquila, Italy
[2] Amrita Center for Cybersecurity Systems and Networks
Amrita School of Engineering
Amritapuri Campus
Amrita Vishwa Vidyapeetham, India

Abstract. meta-models are cardinal assets in Model-Driven Engineering because a diversity of artifacts depend on them, including visual editors. Similar to any other software entity, meta-models are expected to evolve during their life-cycle. Consequently, whenever a meta-model changes, any related artifact must be consistently adapted to preserve its well-formedness, consistency, or intrinsic correctness. Sirius-based editors are no exception. In this paper, we present a study that analyzes the impact of meta-model changes over visual editors based on the Sirius framework. Changes are classified according to their adverse effects in order to provide designers with the possibility to perform an early assessment of the effort needed to restore the editor consistency.

Keywords: Model-Driven Engineering · Co-evolution · Sirius Editors

1 Introduction

It is well recognized that meta-models in Model-Driven Engineering [15] (MDE) represent cardinal assets for any modeling environment. Indeed, the artifacts defined upon a meta-model, whether they be model transformations, code generators, or visual editors, are numerous and at different degrees of dependency [6]. For instance, rules in a model transformation are typically guarded by means of OCL[3] expressions that predicate over the source meta-model and contribute to a relationship typically called domain conformance. Similarly to any other software artifact, meta-models are prone to changes in response to requests for improvements, unforeseen requirements, and new insights emerged from the domain. However, whenever a meta-model changes, any related entity must be consistently adapted for preserving its well-formedness, consistency, or intrinsic correctness [4]. Restoring the consistency between an artifact and the evolved

[3] https://www.omg.org/spec/OCL/

meta-model is a challenging task that may lead to a meta-model lock-in where designers lack confidence when confronted with it. Consequently, designers might decide to procrastinate or even discard meta-model changes to avoid repercussions over the modeling environment.

While the problem of co-evolution of models and meta-models has been extensively investigated (see [10] for a survey) and numerous approaches already exist, e.g., [16, 1, 11], the co-evolution of visual editors and meta-models remains largely unexplored. Despite the significance of maintaining a modeling environment in a consistent state, only one approach on the co-evolution of editors based on GMF[4] has been proposed [7] - to the best of our knowledge. While there is a considerable number of active projects adopting GMF (e.g., Rational Software Architect), recently the use of Sirius[5] is increasingly gaining traction [13]. Sirius is an Eclipse project based on GMF and the Eclipse modeling Framework[6] (EMF) that is increasingly replacing GMF in building modern graphical editors because it requires little programming overhead.

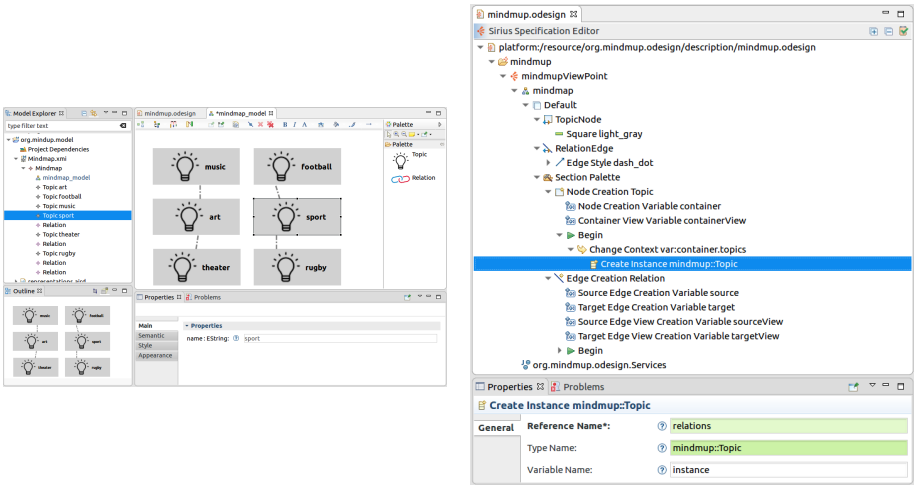
In this paper, a catalog of atomic meta-model changes borrowed from [7] has analyzed in-depth. In particular, each change is discussed and classified according to its impact over Sirius-based editors, and their definition is given by means of the *mapping model* (see Sect. 2). Moreover, to better characterize the change impact, the catalog has been extended by specializing the changes affecting a property in a generic way, for instance 'add property', by distinguishing between attributes and references as they may expose different (and adverse) impact. The possibility of performing an automated consistency restoration is also considered for each meta-model change. The analysis focuses on editors only and does not consider other artifacts that are affected by the changes as well, such as models, transformations, or constraints.

Structure of the paper. The paper is organized as follows. In the next section, a brief introduction to the Sirius framework is given. Section 3 presents a classification of changes according to their impact on the editors. In Sect. 4, each change in the catalog is analyzed and its impact over Sirius-based editors is described. Finally, in Sect. 6 some conclusions are drawn.

2 Sirius

Sirius is an Eclipse project that enables the development of graphical modeling environments by leveraging well-established technologies. Starting from a metamodel (called *domain model* in the Sirius jargon), it allows a model-based specification of visual concrete syntax organized in viewpoints, i.e., models can be authored by means of different notations that suit the needs of various stakeholders. An example editor (for a simple mindmap metamodel) generated with Sirius is represented in Fig. 1a whose perspective includes the graphical editor, the tooling palette, and the property view.

[4] https://www.eclipse.org/gmf-tooling/
[5] https://www.eclipse.org/sirius/
[6] https://www.eclipse.org/modeling/emf/



(a) A simple mindmap editor (b) A mapping model

Fig. 1: The Sirius framework

Sirius allows the conceptual separation between the metamodel and the corresponding representation(s) by means of Viewpoint Specification Models (VSMs)[7], also referred to as mapping models throughout the paper. A mapping model consistently specifies the structure, appearance, and behavior of an editor according to the domain model and it is given in the `.odesign` format. In particular, viewpoints are given in combination with their representation specification that can be grouped in order to provide consistent visual attributions:

- the *representation description* specifies how models are represented, i.e., in the form of diagrams, sequence diagrams, tables, and cross-tables, or trees; user-defined representations can be given also by proper extensions;
- the *representation extensions* that enable the customization of diagrams, i.e., each representation can be endowed with extra functions;
- the *validation rules* are given for validating purposes and for suggesting quick-fixes while editing, and finally
- *Java extensions* define additional functionalities to be associated with editor commands or gestures.

At its very essence, a mapping model is a weaving model describing how each element in the meta-model is represented in the editor. An example of a viewpoint specification is given in Fig. 1b.

Giving an exhaustive description of Sirius goes beyond the scope of this paper. Many aspects have been deliberately ignored (e.g., the way Sirius leaves models unpolluted with the layout and context-specific data by means of `.aird`

[7] https://www.eclipse.org/sirius/doc/specifier/general/Specifying_Viewpoints.html

files) because not pertaining to the discussion. However, the interested reader can refer to the online resources that are given throughout the paper.

3 Change impact classification

As already mentioned, meta-models are expected to evolve during their life-cycle, thus causing possible problems to existing artifacts which refer to the old version of the meta-model and are not consistent to the new version anymore. Classifying changes according to their adverse impact is relevant because it provides designers with the possibility to perform an early assessment of the required effort for restoring the editor consistency. A well-established classification already exists in literature for the co-evolution of models and meta-models [1, 9]: in particular, changes are called *non-breaking* when the conformance of models to the corresponding meta-model is not affected; *breaking and resolvable* when conformance is broken but can be automatically restored; and *breaking and unresolvable* when conformance is broken and its restoration requires human intervention.

Unfortunately, such classification does not apply to the co-evolution of other kinds of artifacts, i.e., each different kind of artifact is depending on one or more meta-models in different ways, for instance domain conformance in transformations is different from model conformance. Therefore, a specific classification for Sirius-based editors has been defined. In this context, we have considered the resilience of an editor as its ability to withstand changes in its environment. Therefore, the classification of changes is given by means of a number of attributes that describe how editors and their specifications are affected as follows.

Editor attributes. The resilience of a Sirius-based editor is explained by its ability to survive meta-model changes and by the degree its expected behavior is still exposed. We characterize the editor resilience by means of the following attributes:

- *non-breaking*: an editor is non-breaking when despite the changes to the meta-model, it is still possible to open and use the editor;
- *complete*: an editor is complete when after a meta-model change, each element in the meta-model has a graphical counterpart within the editor, e.g., as part of the tooling palette;
- *valid*: an editor is valid when it exposes a correct behavior despite the meta-model modifications, i.e., it is still possible to instantiate a model conforming to the new version of the meta-model.

It is worth noting that a non-breaking editor may still be non-complete or non-valid; or that a complete editor may be non-valid because some of the graphical elements in the editor are misplaced, for instance. Moreover, the impact analysis does not consider what happens when a non-breaking editor opens an existing model.

Mapping model attributes. As already described, the mapping model specifies how each modeling element in the meta-model must be visually denoted.

A meta-model change can severely affect the editor specification given by the mapping model stored in the `.odesign` file. The mapping model resilience is characterized by the following attributes:

- *complete*: a mapping model is complete, if after a meta-model change it still contains the graphical denotations for all model elements in the meta-model;
- *valid*: a mapping model is valid, if it consistently defines the graphical denotation for the elements in the meta-model, despite the meta-model changes[8];
- *resolvable*: a mapping model is resolvable, if after meta-model modifications its validity and completeness can be restored by means of an automated procedure.

The attributes given above are used for classifying the changes in the catalog in [7] as described in the next section and summarized in Table 1. It is worth noting that the proposed example covers the concepts of the node, edge, and related palette elements. Thus, the analysis coverage does not comprehend the so-called 'advanced' features as defined in the Sirius documentation[9].

4 How Sirius is affected

The change impact over the Sirius framework is discussed according to the attributes given in the previous section. Additional change have been added with the purpose of specializing modifications in order to be able to distinguish between different editor behaviors, as for instance in the case of 'add property' that has been split into attributes and references. For the sake of reproducibility and ease of reference, all artifacts used in the impact analysis related to a change have been made available as individual branches in a Git repository[10] and named with the same change identifier. A detailed summary of the classification is given in Table 1, where all the atomic changes are given together with their impact on the editor and its corresponding mapping model.

c1 - Add empty, concrete class. The addition of a concrete class in the meta-model is *non-breaking*. The editor is non-complete because it does not expose the new meta-class in the tooling palette. The editor is valid, because the change does not affect its behavior: in EMF it is allowed to instantiate an empty meta-class that does not belong to a container.[11] The corresponding mapping model is non-complete because it does not contain references to the new meta-class, however, it is still valid. The completeness is not resolvable by any automated procedure because it cannot be established ex-ante what is the graphical entity that has to denote the added meta-class. In the following, we have also considered

[8] More formally, a mapping model is valid when the correspondences between the model elements in the meta-model and the mapping model are compatible (in algebraic sense), i.e., if there exists a homomorphism between them.

[9] https://wiki.eclipse.org/Sirius/Tutorials/AdvancedTutorial

[10] https://github.com/MDEGroup/sirius-coevolution

[11] An exception occurs if the added meta-class is the root node, but this is nonsensical as it would not contain any further element.

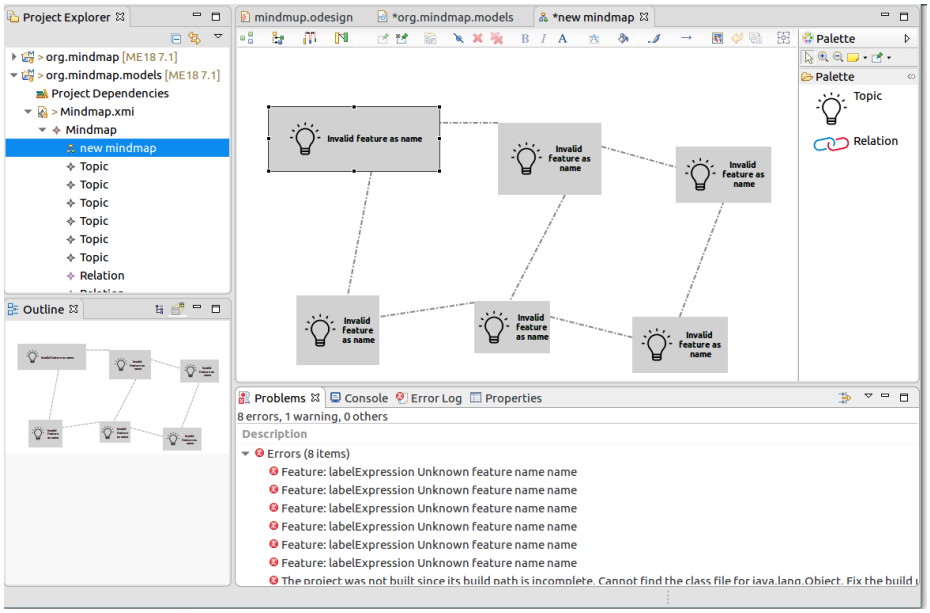


Fig. 2: Delete attribute errors

when the empty, concrete class is associated with an existing one distinguishing between mandatory and optional case.

c1.1 - The new class is 'optional'. The new class is added with an incoming association with multiplicity `0`. The editor is non-breaking. However, it is non-complete since the class is not exposed in the tooling palette (as the added optional reference) because the corresponding descriptions are not-present in the mapping model, which is therefore non-complete. The editor and mapping model is valid. As the change (c1), this is a non-resolvable change.

c1.2 - The new class is 'mandatory'. The new class is added with an incoming mandatory association. The editor is non-breaking, non-complete and non-valid change. Specifically, the non-validity is due to the mandatory association that is not possible to instantiate (because of the editor non-completeness). The mapping model is non-complete is valid and non-resolvable.

c2 - Add empty, abstract class. Adding an empty abstract class does not break the editor that is still complete and valid because an abstract class does not have a graphical denotation in the editor. Analogously, the mapping model is complete and valid. Thus, it is not required to restore its consistency. A variant of this change is considered below.

c3 - Add Specialization. This change specifies new inheritance properties for an existing meta-classes. As to the editor, the change is breaking. The mapping model is non-complete and is non-valid. The change is resolvable.

c4 - Delete Concrete Class. The change breaks the editor. The mapping model is complete but non-valid. The change is resolvable.

c5 - Rename Class. This is a non-breaking change; however, it is non-complete and non-valid. The mapping model is complete but non-valid (it still refers to the previous name of the class). It is resolvable.

c6 - Add Property. *c6.1 - Add Attribute.* This change does not break the editor; however, it is complete and valid (the added attribute can be entered through the property panel). The mapping model is complete and valid.

c6.2 - Add Reference. As in the previous case, the change does not break the editor. Concerning editor completeness and validity, the change has the following impact: no visual counterpart for the reference is provided, thus the editor is non-complete and non-valid (if the reference is mandatory). However, the property panel still allows the specification of the reference. The mapping model is non-complete but valid. The change is not resolvable.

c7 - Delete Property. The change does not break the editor. However, it should be considered non-complete and non-valid because of message errors in the editor and in the 'problems panel' as illustrated in Fig. 2 (that refers to the deletion of an attribute). The mapping model is non-complete and non-valid because of dangling references to the deleted property. The change is resolvable as dangling references can be removed automatically. The impact of the change is very similar for both attributes and references.

c8 - Rename Property.

c8.1 - Rename Attribute. The change does not break the editor. However, because of dangling references the editor is non-complete and non-valid. Error messages are displayed in the editor and in the 'problems panel'. The mapping model is complete but non-valid. The change is resolvable.

c8.2 - Rename Reference. The change does not break the editor. However, because of dangling references the editor is non-complete and non-valid because the renamed references are not displayed. Errors are displayed in the error log. The mapping model is complete but non-valid. The change is resolvable.

c9 - Move Property. The change does not break the editor. However, it should be considered non-complete and non-valid because of message errors in the 'problems panel'. Similarly to the 'add property' change (c6), the property panel allows the property to be updated. The mapping model is complete but non-valid because of dangling references (the behavior is similar to that of a deletion followed by an addition). The change is resolvable as dangling references can be removed automatically. The impact of the change is almost the same for attributes and references.

c10 - Pull up Property. The property is moved from an extended class to a base class. The change does not break the editor that is complete and valid. The mapping model is complete and valid as well. The impact of the change is almost the same for attributes and references.

c11 - Change Property Type.

c11.1 - Change Attribute Type The editor does not break. Moreover is complete and valid. The mapping model is complete and valid as well.

c11.2 - Change Reference Type This is a non-breaking change. However, the editor is non-complete and non-valid as the changed reference is not shown in

id	change	editor			mapping model		
		non breaking	*complete*	*valid*	*complete*	*valid*	*resolvable*
c1	Add empty, concrete class	YES	NO	YES	NO	YES	NO
c1.1	Add empty, optional, concrete class	YES	NO	YES	NO	YES	NO
c1.2	Add empty, mandatory, concrete class	YES	NO	NO	NO	YES	NO
c2	Add empty, abstract class	YES	YES	YES	YES	YES	–
c3	Add specialization	NO	–	–	NO	NO	YES
c4	Delete concrete class	NO	–	–	YES	NO	YES
c5	Rename class	YES	–	–	YES	YES	NO
c6	Add property						
c6.1	Add attribute	YES	YES	YES	YES	YES	–
c6.2	Add reference	YES	NO*	NO*	NO	YES	NO
c7	Delete property	YES	NO	NO	NO	NO	YES
c8	Rename property						
c8.1	Rename Attribute	YES	NO	NO	YES	NO	YES
c8.2	Rename Reference	YES	NO	NO	YES	NO	YES
c9	Move property	YES	NO*	NO*	YES	NO	NO
c10	Pull up property	YES	YES	YES	YES	YES	–
c11	Change property type						
c11.1	Change attribute type	YES	YES	YES	YES	YES	–
c11.2	Change reference type	YES	NO	NO	NO	NO	YES

Table 1: Summary of the change impact (- = not applicable, * = involves property view)

the editor (not even in the property panel). The mapping model is non-complete and non-valid (if the reference is mandatory). The change is resolvable.

5 Related Work

The problem of co-evolution [5] has been extensively investigated for models [10]. Existing approaches can be distinguished in *state-based* approaches [1] and *operation-based* ones [11, 14, 17]: the former derive migration procedures from the meta-model differences, whereas the latter programmatically specify how models must be co-adapted.

Different approaches have been proposed for the co-evolution of model transformations: in [8] higher-order transformations generate a pre-packaged solution to be customized according to the designer's needs, predefined migration actions are proposed in [11, 12], whereas in [18] mapping operators are used. All the

mentioned approaches are considered deterministic, i.e., they identify one transformation adaptation, typically based on some heuristics, also in those cases where multiple valid alternatives are possible. The problem of non-deterministic transformation adaptation has been discussed and addressed in [3], where alternative solutions have been combined together by means of variability models expressing the different valid migration policies. Designers can then explore the solution space in order to choose the right migration according to criteria that normally cannot be expressed easily in procedural terms.

In [6] the problem of co-evolution is discussed under a different light, i.e., that available approaches focus on one specific kind of migration (for instance the migration of models, transformation, or the adaptation of editors). As a consequence, if the whole modeling environment must be co-evolved, then designers must become familiar with a diversity of techniques, making the overall endeavor difficult and prone to mistakes. Thus, the authors advocate the need for a holistic approach that let designers express the rationale behind meta-model evolution to be used for the migration of the whole environment.

Finally, the problem of the co-evolution of GMF editors is addressed in [7]. The work analyzes the dependencies between a meta-model and all the models defining a GMF editor, classifies the changes according to the impact, and proposes a (semi) automated procedure for its adaptation. Clearly, our classification presents similarities with the approach in [7]. However, the classification proposed for GMF editors is based on the final effect on the editor rather than understanding the exact dependencies among the meta-model and the editor specification, which is the main focus of this paper. Moreover, in GMF the editor specification is different from that in Sirius, therefore it is not easy to extend the results in [7] to Sirius.

While the work in [7] is (to the best of our knowledge) the only systematic classification of the metamodel change impact, a number of platforms, including MetaEdit+[12] and Whole Platform[13], provide some support to coupled evolution by allowing the combination and extension of languages and the corresponding environments [2]. However, an analysis of the change impact for them (comparable to that in [7]) is not available.

It is worth noting that the corpus of research about coupled evolution is extensive. In this paper, we discussed the most prominent works about the co-evolution of the most important artifacts.

6 Conclusions

The Sirius framework is an increasingly adopted tool for the definition of visual modeling environments built by leveraging EMF meta-models. Similarly to what happens with models (and other artifacts), changes to a meta-model typically reverberate through the editor at different degrees of severity putting the overall project in jeopardy. In this paper, we have analyzed and classified a catalog of

[12] https://www.metacase.com/mep/
[13] http://whole.sourceforge.net

atomic changes borrowed from [7] in accordance with their impact over Sirius editors. For each change, it has been observed whether the editor fails to open and whether it exposes correct and complete behavior. Moreover, we analyzed the impact on the editor definition, i.e., the mapping model, to understand whether the restoration procedure can be automated or if user intervention is needed. The work in this paper is preliminary to the analysis of more extended and composite refactoring patterns and to the definition of (semi) automated procedures for the consistency restoration. Besides more extended changes, future work includes also the analysis of changes operated on meta-model OCL invariants and how they affect editor resilience.

Acknowledgments. The research described in this paper has been partially supported by the CROSSMINER Project, EU Horizon 2020 Research and Innovation Programme, grant agreement No. 732223.

References

1. Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A.: Automating co-evolution in model-driven engineering. In: Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE. pp. 222–231. IEEE (2008)
2. van deStorm, T., Erdweg, S., Voelter, M., Boersma, M., Bosman, R., Cook, W., Gerritsen, A., Hulshout, A., Kelly, S., Loh, A., et al.: The state of the art in language workbenches. conclusions from the language workbench challenge (2013)
3. Di Ruscio, D., Etzlstorfer, J., Iovino, L., Pierantonio, A., Schwinger, W.: A feature-based approach for variability exploration and resolution in model transformation migration. In: European Conference on Modelling Foundations and Applications. pp. 71–89. Springer (2017)
4. Di Ruscio, D., Iovino, L., Pierantonio, A.: What is needed for managing co-evolution in MDE? In: Proceedings of the 2nd International Workshop on Model Comparison in Practice. pp. 30–38. ACM (2011)
5. Di Ruscio, D., Iovino, L., Pierantonio, A.: Coupled evolution in model-driven engineering. IEEE software **29**(6), 78–84 (2012)
6. Di Ruscio, D., Iovino, L., Pierantonio, A.: Evolutionary togetherness: how to manage coupled evolution in metamodeling ecosystems. In: International Conference on Graph Transformation. pp. 20–37. Springer (2012)
7. Di Ruscio, D., Lämmel, R., Pierantonio, A.: Automated Co-evolution of GMF editor models. In: International Conference on Software Language Engineering. pp. 143–162. Springer (2010)
8. García, J., Diaz, O., Azanza, M.: Model transformation co-evolution: A semi-automatic approach. In: International Conference on Software Language Engineering. pp. 144–163. Springer (2012)
9. Gruschko, B., Kolovos, D., Paige, R.: Towards synchronizing models with evolving metamodels. In: Proceedings of the International Workshop on Model-Driven Software Evolution. pp. 3–1. IEEE (2007)
10. Hebig, R., Khelladi, D.E., Bendraou, R.: Approaches to co-evolution of metamodels and models: A survey. IEEE Transactions on Software Engineering **43**(5), 396–414 (2017)
11. Herrmannsdoerfer, M.: Cope–a workbench for the coupled evolution of metamodels and models. In: International Conference on Software Language Engineering. pp. 286–295. Springer (2010)

12. Kruse, S.: On the use of operators for the co-evolution of metamodels and transformations. In: International Workshop on Models and Evolution (2011)
13. Liebenberg, M., Roßmaier, K., Lakemeyer, G.: An istar 2.0 editor based on the eclipse modelling framework
14. Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.: Model migration with epsilon flock. In: International Conference on Theory and Practice of Model Transformations. pp. 184–198. Springer (2010)
15. Schmidt, D.C.: Model-driven engineering. IEEE Computer **39**(2), 25 (2006)
16. Wachsmuth, G.: Metamodel adaptation and model co-adaptation. In: European Conference on Object-Oriented Programming. pp. 600–624. Springer (2007)
17. Wagelaar, D., Iovino, L., Di Ruscio, D., Pierantonio, A.: Translational semantics of a co-evolution specific language with the emf transformation virtual machine. In: International Conference on Theory and Practice of Model Transformations. pp. 192–207. Springer (2012)
18. Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schoenboeck, J., Schwinger, W.: Surviving the heterogeneity jungle with composite mapping operators. In: International Conference on Theory and Practice of Model Transformations. pp. 260–275. Springer (2010)