Improving Trace-Based Propagation of Feature Annotations in Model Transformations

Sandra Greiner and Bernhard Westfechtel

University of Bayreuth, Universitaetsstrasse 30, 95440 Bayreuth, Germany
{Sandra1.Greiner,Bernhard.Westfechtel}@uni-bayreuth.de

Abstract. In annotative approaches to model-driven software product line engineering variability annotations express in which variants model elements are included. Although (single-variant) model transformations (SVMTs) are mature by now, they cannot process these annotations. Considering them calls for multi-variant model transformations (MVMTs) which ideally reuse existing transformations. In trace-based approaches annotations are propagated by exploiting the information of traces written alongside the transformation. Depending on the granularity of the trace, different strategies can be applied. For instance, fine-granular traces record not only all source and target elements of a rule application but list context elements. The latter have been created in previous rule applications and are necessary for creating the new elements. In this paper we demonstrate an example in which a propagation based on context elements fails. Contrastingly, we propose to propagate only annotations of 1:1 mappings and to, thereafter, calculate the annotations for elements still missing one. Moreover, we introduce a new evaluation criterion based on incremental transformations, allowing to take the multi-variant context into account.

1 Introduction

In *model-driven software engineering* (MDSE) [17] model transformations are the key technology to transform a source model (input) into a different representation, the target model (output). While *unidirectional* transformations convert a source into a target model, *bidirectional* transformations also consider the backward direction in the same specification. A *batch* transformation creates the target model completely anew whereas an *incremental* transformation applies changes made to the source model after a previous execution. *In-place* transformations modify the input model itself, which, hence, is the target model. Such transformation is always *endogenous* since the target metamodel remains the same. In contrast, in *out-place* transformations a different physical representation is created or modified, mostly in *exogenous* transformations where the output metamodel differs from the input one. Often *traces* record the source and their corresponding target elements for each rule application. Triple graph grammars (TGG) [15] and comparable approaches map exactly one source element to one target element in a *correspondence graphs*.

Furthermore, in *model-driven software product line engineering (MDPLE)* [6] models express a set of related products based on the principle of *organized reuse*. The *platform* captures the complete functionality w.r.t. its commonalities and differences. Typically, *feature models* [11] state the distinguishing factors as *features*. A product is derived by *filtering* the platform by a *feature configuration*, which provides each feature with a (de-)selected state. MDPLE heavily relies on model transformations in various development stages in order to automatically transfer a model to a new representation, e.g., in refactoring tasks or when transforming in between design and realization models.

In *annotative* approaches to MDPLE [2] model elements are associated with presence conditions which are boolean expressions over the features, defining, in which variants a model element is visible. We refer to them as *annotations*. Annotated platform models typically capture the whole product line, i.e., all its variants. Therefore, we call such models *multi-variant models (MVMs)*. Transforming an annotated MVM in aforementioned transformations results in a target MVM missing annotations which are not recognized in *single-variant model transformations (SVMTs)*. A transformation considering annotations is called a *multi-variant model transformation (MVMT)*. Some approaches already realize MVMTs, mostly by *reusing* SVMTs. *Lifting* [14] and approaches with variability-based rules [18] exploit the contents of (graph) transformation specifications. More generally, the SVMT could be executed as a black-box. Thus, annotations need to be propagated alongside the SVMT, usually, *a posteriori*. *Trace-based* approaches [22,7], primarily realize the propagation based on trace information.

In this paper we contribute a novel trace-based approach to MVMTs solely using 1:1 mappings. In a post-processing step annotations for target elements still missing one are determined from dependencies inside the target model. Furthermore, we introduce an *incremental commutativity* criterion to evaluate the correctness of the annotated target MVM, allowing to consider the multi-variant context in the evaluation. Applying the procedure on an example transformation, in which other approaches fail, achieves the desired behavior.

In the next section we delimit our MVMT approach from existing ones. Then, we describe an example violating commutativity and provide a solution to the problem. From the findings future work is conducted.

2 Background

2.1 MVMT Realization Approaches

Different strategies for realizing MVMTs can be roughly categorized in *black-box* and *white-box* solutions, requiring *no* or *all* internals of the SVMT, respectively.

The lifting algorithm [14] and combining it with variability-based rules [18] are categorized as white-box solutions. Lifting executes (graph) transformation rules with multi-variant semantics which requires to enumerate all rules for lifting the product line. Although the algorithm is defined for in-place graph transformations, it was extended to out-place transformations with a graph-like DSL [5].

The proposal in [18] is restricted by the same properties but is executed more efficiently. The formalism based on category theory presented in [19] relies on the contents of the transformation as well. Alternatively, in [16] ATL rules [10] are extended to support variability and converted to a normal ATL transformation in higher order transformations (HOT).

Contrastingly, *black-box* approaches only know the input and output of the transformation. The SVMT remains exactly the same. Annotations can, then, be propagated by applying different strategies: In [4] annotations are assigned by manually defining corresponding elements of the source and target metamodel. Thus, for each new transformation a mapping definition must be provided. In contrast, in [8] the authors "inject" annotations as preprocessor directives with a generic aspect in *Xpand* [12] transformations. This approach works independently of the transformation specification and the metamodels.

2.2 Trace-based Propagation

Traces are another source of information for propagating annotations. A trace-based solution [22] is rather categorized as *grey-box* approach since knowing the source and target elements of rule applications implicitly exploits transformation contents. We can distinguish *incomplete*, *generation-complete* and *complete* traces. *Incomplete traces* record only 1:1 mappings. If more than one element is created, only the first (most important) target element is kept. Such "trace" is similar to *correspondence graphs* in TGGs. Languages and tools based on TGGs, e.g., eMoflon [13] or BXtend [3], provide (at least) this level of knowledge. Contrastingly, a *generation-complete trace* stores all source and all target elements associated with a rule application, like the trace written by the ATL/EMFTVM [21]. *Complete traces* further partition the elements of the target model: an element already existing in the target model upon the (later) execution of a rule is called *context element*. Such elements may be needed to create *new* target elements and are, thus, distinguished in the trace. The tool medini QVT [9] stores complete trace as well as eMoflon in a *protocol* besides the correspondence graph.

In a simplified way Algorithm 1 describes how to propagate annotations based on a complete trace [22]. For each trace element, the annotation propagated to its target elements (l. 8-9) is a conjunction (l. 6) of the annotations of its source (l. 4) and its context (l. 5) elements.

For validating the correctness of the propagation, a *commutativity* criterion (Figure 1) is postulated for annotative approaches in [14,8] and for feature-oriented ones in [20]. It checks whether the result m_t from transforming an annotated source MVM m_s with an MVMT is correctly annotated. The MVMT consists of the reused SVMT (t_{sv}) and the annotation propagation. Filtering m_s by a feature configuration and transforming the result m'_s with t_{sv} creates a target product m''_t which should be equal to the model m'_t being filtered from m_t by the same configuration. This property must hold for all valid feature configurations.

In [22] it is shown that trace-based propagation achieves commutativity for specific transformations. Next, we demonstrate an example violating the postu-

Algorithm 1 Trace-based propagation of annotations.

```
1: procedure PROPAGATE(te)
2:     in te                                   ▷ Trace composed of trace elements te
3:     for te ∈ te do                          ▷ Process all trace elements sequentially
4:         var ann_s = te.getSrcAnns()         ▷ Annotations of source elements
5:         var ann_c = te.getCntxtAnns()       ▷ Annotations of context elements
6:         var ann_t := ⋀ann_s ∧ ⋀ann_c         ▷ Annotation for target elements
7:
8:         for e_t ∈ te.getTrgElements() do    ▷ Process all target elements
9:             e_t.setTargetAnnotation(ann_t)  ▷ Annotate the target element
```

lated computational model and, thus, failing to commute. For that reason, we state the following questions:

1. To which granularity can the information of traces be helpful? Are context elements (always) decisive for the success of the MVMT?
2. Are batch transformations appropriate for evaluating commutativity?

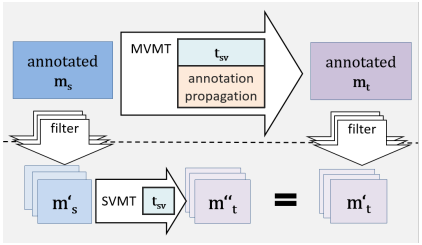

Fig. 1. Commutativity of MVMTs.

3 Example

3.1 Setup

The following scenario is based on a benchmark evaluating bidirectional (and incremental) model transformations [1]. Since we like to rather demonstrate key ideas than to provide a complete evaluation, the models are kept small.

Let us migrate a database storing persons to one storing families. As depicted on the left hand side of Figure 2, named persons (composed of the last name followed by the first name) are registered as male or female. In contrast, the family database records families by their last names and a family references its members according to their role (mum, dad, etc.) inside the family.

The family database is created from the persons database as follows (t_{sv}): Each person is inserted as member in a family with the respective last name.

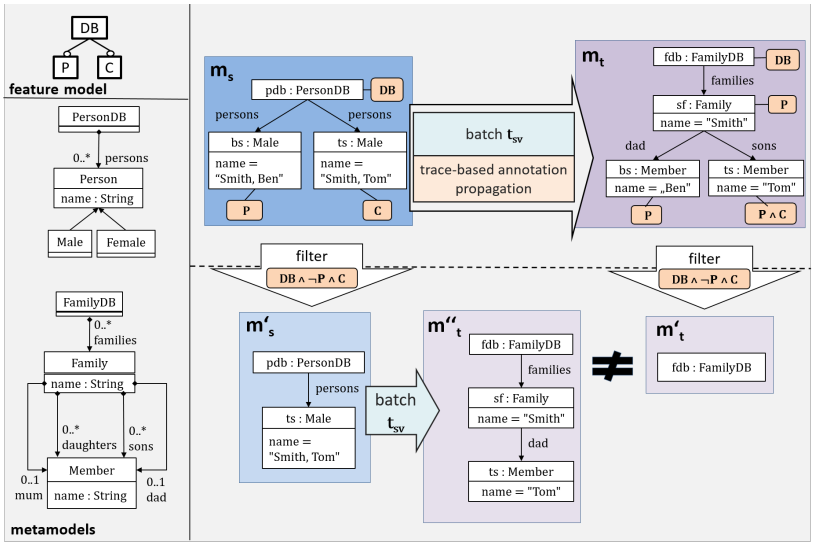

Fig. 2. Metamodels (left) on which the transformation scenario (right) is based.

If there is no appropriate family, a Family element is created and the person is inserted as parent. If the family exists but the parent role is not occupied, the member will also be a parent. If both parents exist, the member becomes a child.

3.2 Multi-Variant Transformation

Now let us consider the example in the context of a software product line. The feature model (top left of Figure 2) consists of the optional features parent (P) and children (C). As depicted on the right of Figure 2, on instance level, the person model consists of two men with the same last name 'Smith'. Ben is annotated with the feature P since he is the father of the family, and Tom should be a child (feature C). Transforming this model with the t_{sv} described above, creates a Family Smith with a member Ben as father and a member Tom as son. A complete trace would record three elements as listed in Table 1. Please note: The last row states the rule application with Tom as source element. It lists the family as context element because it already exists and is necessary for inserting the member Tom as a child.

The trace-based propagation (Algorithm 1) as explained in Section 2.2 annotates the target elements as depicted in Figure 2: The member Ben and the family receive Ben's annotation as they are both recorded as its target elements. Secondly, Tom's annotation is the conjunction of the annotation of its source element (C) and of its context element (P), the Family.

Finally, products are derived and commutativity is evaluated. Deriving, for instance, a database containing only children, we filter both, m_s and m_t, by the feature configuration $DB \wedge \neg P \wedge C$. The derived source product m'_s includes

Table 1. Trace contents.

rule application	source elements	target elements	context elements
PDb2FDB	pdb : PersonDB	fdb:FamilyDB	–
Male2Member	bs : Male	sf : Family, bs: Member	–
Male2Member	ts : Male	ts : Member	sf : Family

the elements as expected: only the male Tom remains. Filtering m_t, however, results in a product m'_t consisting of the database only. Although this is not the desired semantic result, m'_t could still be correct w.r.t. the commutativity criterion (in the case t_{sv} creates an equivalent model m''_t with m'_s as input). Hence, for verifying commutativity, we execute t_{sv} with the product m'_s as input. The output, m''_t, however, is semantically incorrect, too: Tom becomes the father of the family as from the transformation's point of view the family and a parent are missing. Thus, Tom triggers the creation of the family and is added as father. Since m'_t and m''_t are not equivalent, commutativity is violated.

Summing it up, (semantic) misbehavior appears at three locations:

1. The annotation attached to the Family element in m_t is wrong since the family must exist as soon as any member is present.
2. The annotation of Tom in m_t might be wrong. Tom should be part of a target product whenever its counterpart exists in a source product.
3. Applying t_{sv} on the source product m'_s creates a semantically wrong target m''_t.

4 Solution Approach

4.1 Procedure

The first two aforementioned problems tackle the propagation of annotations. The new procedure should adhere to Algorithm 2, assuming all metamodels to be instances of the Ecore meta-metamodel (i.e., directed, acyclic graphs). At first, only the annotations of 1:1 mappings are copied from the source to the (main) target element[1]. In the *post-processing* phase (l. 6-10), firstly, all elements missing annotations are collected top-down. For each unannotated element e the annotations of elements required for its existence (l. 7) and the ones requiring e's existence (l. 8) are determined. The contents of the sets are discussed in Section 4.3. Then, the annotation is calculated as in line 9 of Algorithm 2, i.e., e is present if all the elements it needs are present, and, if at least one of the elements needing e's presence exists. Finally, the annotation is set on e (l. 10).

For evaluating commutativity, batch transforming the source products results in an unintended target model (3rd problem). Therefore, we propose an *incremental commutativity* criterion: Firstly, the MVMT is executed as before

[1] Copying the annotation of the source to its corresponding target is trivially correct as the target element should exist when its corresponding source element exists.

Algorithm 2 Improved propagation of annotations.

```
1: procedure PropagateImproved(te, m_t)
2:     in te                          ▷ Trace composed of single trace elements te
3:     in m_t                         ▷ Multi-variant target model
4:
5:     propagateBasicMappings(te)
6:     for e ∈ m_t.getUnannotatedElements() do
           ▷ Process target elements without annotation in topological order
7:         var ann_needed = m_t.getAnnsOfNeeded(e)
               ▷ Annotation of elements needed for the existence of e
8:         var ann_needing = m_t.getAnnsOfNeeding(e)
               ▷ Annotation of elements needing the existence of e
9:         var ann_t := ⋀ ann_needed ∨ ⋁ ann_needing
10:        e.setTargetAnnotation(ann_t)          ▷ Annotate the target
```

by reusing the single-variant (batch) transformation t_{sv}. As opposed to the commutativity criterion postulated in Section 2.2, source products (m'_s) are transformed incrementally. Thus, decisions made in the multi-variant context, e.g., a person's role in the family, are taken into account.

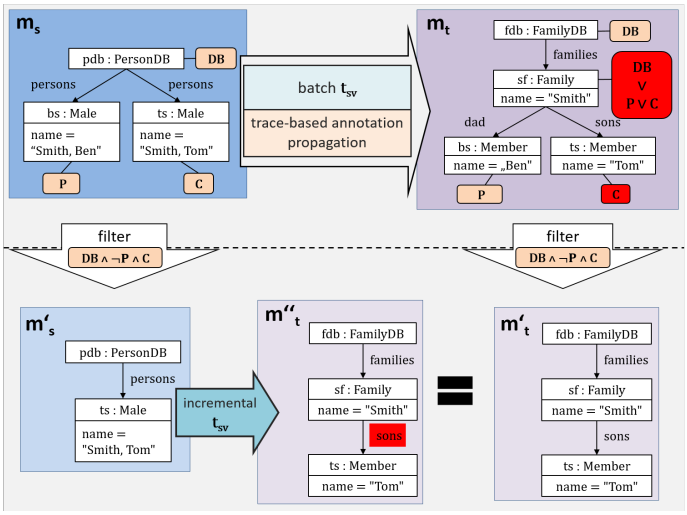

Fig. 3. Commutativity based on the improved propagation and evaluation.

4.2 Results

Figure 3 depicts the improved example. Firstly, t_{sv} is (batch) transformed with the source MVM m_s producing the target MVM (without annotations) m_t. Next,

the annotations of the basic mappings are applied. Tom and Ben alike receive only the annotations of their corresponding source elements. Thereafter, the Family still expects an annotation being calculated in the post-processing step. The element receives the presence condition $DB \vee (P \vee C)$. It is composed of the annotations of the elements needed for its existence, the database (DB), and of the elements requiring its existence (Ben and Tom) combined in a disjunction.

Evaluating commutativity by incrementally executing t_{sv}, Ben is removed and Tom's role remains the same in m_t''. Moreover, filtering the annotated target model m_t by the same feature configuration behaves well: the family and Tom are not removed anymore. Thus, the example now commutes.

Accordingly, the questions stated in Section 2.2 may be answered as follows:

1. In our example context elements hinder the correct annotation of target elements. Context elements may change when switching from the multi-variant to the single-variant context. Contrastingly, 1:1 mappings are trivially correct and do not require fine-granular trace information.
2. Commutativity should be evaluated based on incrementally transforming source products considering, thus, decisions of the multi-variant context.

4.3 Discussion

Our solution is the first step to an automatic trace-based propagation supporting the most general form of (an incomplete) trace. The two-step procedure is still trimmed to the example needing further evaluation.

The key difficulty has been moved to the post-processing step. It seems to be reasonable to combine annotations of required and requiring elements as stated in Algorithm 2. However, the definition is silent on the elements belonging to the respective sets. They must be determined from the output (meta-)model and depend on the granularity of annotating model elements. For instance, required elements subsume at least the container but might also be the types of all referenced typed elements, supertypes and operation parameters in the case complete objects are annotated. Requiring elements are at least all contained objects. Moreover, if too much elements are unannotated after the basic propagation, calculating the annotation automatically may become impossible.

Finally, the incremental evaluation is beneficial when elements are deleted upon filtering. However, if a filtered source model (m_s') adds a new element or changes it differently than in the MVMT, this change will not be part of the MVM m_t and, hence, of m_t'[2]. Moreover, not all, though many, languages support incremental transformations in an unconstrained way.

5 Summary

To sum it up, we present a novel approach to circumvent problems arising in trace-based approaches to MVMT. Instead of exploiting fine-granular traces,

[2] For a correct MVM, changes resulting from transforming source products should be integrated into m_t, which may violate single-variant semantics.

which are rarely present, we only propagate annotations of 1:1 mappings. They form part of every trace and copying them is definitely correct. Missing annotations are calculated based on already present annotations and dependencies inside the model. For evaluating commutativity we propose to perform incremental transformations on product level taking into account the multi-variant context. Applying the approach on initial examples achieves the desired result.

As next steps, we like to generalize the approach and provide extensive evaluation. In particular, the post-processing should be investigated in greater detail.

References

1. Anjorin, A., Diskin, Z., Jouault, F., Ko, H., Leblebici, E., Westfechtel, B.: BenchmarX Reloaded: A Practical Benchmark Framework for Bidirectional Transformations. In: Proceedings of the 6th International Workshop on Bidirectional Transformations co-located with The European Joint Conferences on Theory and Practice of Software, BX@ETAPS 2017, Uppsala, Sweden, April 29, 2017. pp. 15–30 (2017)
2. Apel, S., Janda, F., Trujillo, S., Kästner, C.: Model Superimposition in Software Product Lines. In: Theory and Practice of Model Transformations, Second International Conference, ICMT 2009, Zurich, Switzerland, June 29-30, 2009. Proceedings. pp. 4–19 (2009). https://doi.org/10.1007/978-3-642-02408-5_2
3. Buchmann, T.: BXtend - A Framework for (Bidirectional) Incremental Model Transformations. In: Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira - Portugal, January 22-24, 2018. pp. 336–345 (2018). https://doi.org/10.5220/0006563503360345
4. Buchmann, T., Greiner, S.: Managing Variability in Models and Derived Artefacts in Model-driven Software Product Lines. In: Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira - Portugal, January 22-24, 2018. pp. 326–335 (2018). https://doi.org/10.5220/0006563403260335
5. Famelis, M., Lucio, L., Selim, G.M.K., Sandro, A.D., Salay, R., Chechik, M., Cordy, J.R., Dingel, J., Vangheluwe, H., Ramesh, S.: Migrating automotive product lines: A case study. In: Theory and Practice of Model Transformations - 8th International Conference, ICMT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-21, 2015. Proceedings. pp. 82–97 (2015). https://doi.org/10.1007/978-3-319-21155-8_7
6. Gomaa, H.: Designing software product lines with UML 2.0: From use cases to pattern-based software architectures. In: Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, Maryland, USA, August 21-24, 2006, Proceedings. p. 218 (2006). https://doi.org/10.1109/SPLINE.2006.1691600
7. Greiner, S., Schwägerl, F., Westfechtel, B.: Realizing multi-variant model transformations on top of reused ATL specifications. In: Pires, L.F., Hammoudi, S., Selic, B. (eds.) Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017). pp. 362–373. SCITEPRESS Science and Technology Publications, Portugal, Porto, Portugal (February 2017). https://doi.org/10.5220/0006137803620373
8. Greiner, S., Westfechtel, B.: Generating multi-variant java source code using generic aspects. In: Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD

2018, Funchal, Madeira - Portugal, January 22-24, 2018. pp. 36–47 (2018). https://doi.org/10.5220/0006536700360047
9. ikv++ technologies: medini QVT. ikv++ technologies (2018), http://projects.ikv.de/qvt
10. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comput. Program. **72**(1-2), 31–39 (2008). https://doi.org/10.1016/j.scico.2007.08.002
11. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, Carnegie-Mellon University, Software Engineering Institute (Nov 1990)
12. Klatt, B.: Xpand: A closer look at the model2text transformation language. Language **10**(16), 2008 (2007)
13. Leblebici, E., Anjorin, A., Schürr, A.: Developing emoflon with emoflon. In: Theory and Practice of Model Transformations - 7th International Conference, ICMT 2014, Held as Part of STAF 2014, York, UK, July 21-22, 2014. Proceedings. pp. 138–145 (2014). https://doi.org/10.1007/978-3-319-08789-4_10
14. Salay, R., Famelis, M., Rubin, J., Sandro, A.D., Chechik, M.: Lifting model transformations to product lines. In: 36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014. pp. 117–128 (2014). https://doi.org/10.1145/2568225.2568267
15. Schürr, A.: Specification of graph translators with triple graph grammars. In: Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94, Herrsching, Germany, June 16-18, 1994, Proceedings. pp. 151–163 (1994). https://doi.org/10.1007/3-540-59071-4_45
16. Sijtema, M.: Introducing variability rules in atl for managing variability in mde-based product lines. Proc. of MtATL **10**, 39–49 (2010)
17. Stahl, T., Völter, M., Bettin, J., Haase, A., Helsen, S.: Model-driven software development - technology, engineering, management. Pitman (2006)
18. Strüber, D., Rubin, J., Arendt, T., Chechik, M., Taentzer, G., Plöger, J.: Variability-based model transformation: formal foundation and application. Formal Aspects of Computing **30**(1), 133–162 (Jan 2018). https://doi.org/10.1007/s00165-017-0441-3
19. Taentzer, G., Salay, R., Strüber, D., Chechik, M.: Transformations of software product lines: A generalizing framework based on category theory. In: 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, Austin, TX, USA, September 17-22, 2017. pp. 101–111 (2017). https://doi.org/10.1109/MODELS.2017.22
20. Trujillo, S., Batory, D.S., Díaz, O.: Feature oriented model driven development: A case study for portlets. In: 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007. pp. 44–53 (2007). https://doi.org/10.1109/ICSE.2007.36, `https://doi.org/10.1109/ICSE.2007.36`
21. Wagelaar, D., Iovino, L., Ruscio, D.D., Pierantonio, A.: Translational semantics of a co-evolution specific language with the EMF transformation virtual machine. In: Theory and Practice of Model Transformations - 5th International Conference, ICMT 2012, Prague, Czech Republic, May 28-29, 2012. Proceedings. pp. 192–207 (2012). https://doi.org/10.1007/978-3-642-30476-7_13
22. Westfechtel, B., Greiner, S.: From Single- to Multi-Variant Model Transformations: Trace-Based Propagation of Variability Annotations. In: 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018. p. accepted for publication