Change Propagation-based and Composition-based Co-evolution of Transformations with Evolving Metamodels

Djamel Eddine Khelladi, Roland Kretschmer, Alexander Egyed

Change Propagation-based and Composition-based Co-evolution of Transformations with Evolving Metamodels

Djamel Eddine Khelladi, Roland Kretschmer, Alexander Egyed
Institute for Software Systems Engineering, Johannes Kepler University Linz, Austria.
{djamel_eddine.khelladi,roland.kretschmer,alexander.egyed}@jku.at

ABSTRACT

Transformations constitute significant key components of an automated model-driven engineering solution. As metamodels evolve, model transformations may need to be co-evolved accordingly. A conducted experiment on transformations' co-evolution highlighted the existing gap in the literature where only limited few co-evolution scenarios are covered without supporting alternatives that occur in practice. To make matters worse, when a developer needs to drift apart from the proposed co-evolution, no automatic support is provided to the developer. This paper first proposes a change propagation-based co-evolution of transformations. The premise is that knowledge of the metamodel evolution can be propagated by means of resolutions to drive the transformation co-evolution. To deal with particular cases where developers must drift from the proposed resolutions, we introduce a composition-based mechanism that allows developers to compose resolutions meeting their needs. Our work is evaluated on 14 case studies consisting in original and evolved metamodels and ETL Epsilon transformations. A comparison of our co-evolved transformations with the 14 versioned ones showed the usefulness of our approach that reached an average 96% of correct co-evolution. On three other case studies, our composition-based co-evolution showed to be useful to eight developers in selecting resolutions that best meet their needs. Among the applied resolutions, four developers applied six resolutions that were the direct result of a composition.

ACM Reference Format:
Djamel Eddine Khelladi, Roland Kretschmer, Alexander Egyed. 2018. Change Propagation-based and Composition-based Co-evolution of Transformations with Evolving Metamodels. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3239372.3239380

1 INTRODUCTION

Model-Driven Engineering (MDE) has proven to be effective in the development and maintenance of large scale and embedded systems [21, 22]. Today modeling languages play a significant role in all phases of development processes [38]. The very foundation of modeling languages are their metamodels [22]. The metamodels define the vocabulary and syntax construction of the language. Metamodels are the foundation for the creation of model instances. However, it is often overlooked that metamodels also serve for the creation of other artifacts such as model transformations.

Model transformations play a significant role in *MDE* [32, 36] by specifying and automating the transformation of source models to either target models, i.e., model-to-model transformation (e.g., UML [33] to a database model) or to target texts, i.e., model-to-text transformation (e.g., UML to LaTeXor HTML documentation).

One of the foremost challenges to deal with in *MDE* is evolution of metamodels and its impact on artifacts that use metamodels as a foundation. As a result of metamodel evolution, transformations may fail to execute. Hence, they may need to be co-evolved accordingly to function properly. Knowing that model transformations are used for different activities, such as code generation [16], model refactoring [31] and migration [41]. It is essential to efficiently co-evolve the transformations whenever the metamodel evolves.

An experiment on transformation co-evolution [28] highlighted that only limited few co-evolution scenarios are covered in the literature and no alternative co-evolutions are supported. For every impacted transformation, only one co-evolution is enforced. Hence, the performed co-evolution cannot meet the different developers needs. To make matters worse, when a developer has to drift apart from the proposed co-evolutions, no automatic support is provided. Thus, limiting the usage of existing approaches in practice.

This paper first proposes a change propagation-based co-evolution of model transformations. Change propagation showed to be efficient in many domains, such as in the context of source code or models co-evolution [5, 6, 13]. In this paper, change propagation leverages on the knowledge provided by the metamodel changes during evolution. We propagate those changes on the transformations. For example, knowing that a property *p* in the metamodel is deleted or renamed, it is likely that the used property *p* will also be respectively deleted or renamed in the transformations. We thus propose a catalog of resolutions that is used as a basis for our change propagation. However, in rare cases, the proposed co-evolution may not meet the developer needs. For example, instead of deleting the property *p*, the developer may need to replace it by another property *p'*, possibly from another class. Thus, We also propose a composition-based mechanism that allows developers to freely select resolutions from our catalog. But also to compose resolutions into a new one. Both of our co-evolution modes are applied per impacted transformation.

Our change propagation is evaluated on 14 case studies consisting of several versions of metamodels and ETL Epsilon transformations. A comparison of our co-evolved transformations with the 14 versioned transformations shows that our approach supports useful alternative co-evolutions that developers have manually performed.

MODELS '18, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-4949-9/18/10…$15.00
https://doi.org/10.1145/3239372.3239380

Within our case studies, 83 resolutions were applied on the impacted transformations leading to an average of 96% correctness of co-evolution. Our composition-based co-evolution is evaluated on additional three case studies with eight developers. The observed results showed its usefulness to the developers in freely selecting resolutions that best fist their needs. Among the applied resolutions, four developers applied six resolutions that were the direct result of a composition.

2 MOTIVATING EXAMPLE

To motivate our work, this section introduces an example of model transformations taken from [2, 3][1]. It illustrates a model transformation from the SimplePDL process modeling language to the PetriNet modeling language. Figures 1 and 2 depict an excerpt of the SimplePDL and the PetriNet metamodels as defined in [2, 3].

SimplePDL is a language for specifying processes composed of a set of *work definitions* representing the activities to be performed during development. *Work definitions* are executed in a defined *work sequence* with an ordering constraint specified by a *link Type* between two work definitions. *Resources* to perform one work definition (e.g., designer, computer, server) can be defined in the process as well as their *parameters*. PetriNet models are composed of *nodes* which can be *places* or *transitions*. *Nodes* are linked together by *arcs* that can either be normal or read-arcs. Both *places* and *arcs* contains tokens that are consumed or produced during the petrinet model execution. Listing 1 shows three simple transformation rules among the specified ones in [2, 3]. The first transforms a process to a petri net, the second transforms a resource to a place, and the last transforms a parameter to arcs.

Consider that the SimplePDL metamodel evolves with the following changes: 1) Delete the properties *minTime* and *maxTime* in Process (they can be deduced from the process's work definitions), 2) rename the property *name* to *URI* in Resource, and 3) Move the property *isNormal* from Resource to Parameter.

The first arising issue is that some existing editors (e.g., ATL[2] and ETL[3] editors) do not show the impact of metamodel changes on transformations, i.e., do not highlight errors. For example, even though the the properties *minTime* and *maxTime* are deleted no errors are raised by the transformation editors. Only once executed the errors are detected. Therefore, it is essential to support an impact analysis to identify the impacted transformations [23]. Alternatively, static checker can be used, such as *anATLyzer* [4].

The three above changes are known to be impacting transformations from [29]. The impacted parts of the three transformations are underlined (in red) in Listing 1 and their co-evolution is underlined (in green) in Listing 2. For some impacts, the co-evolution can be straightforward. For example, the renamed property *name* to *URI* in class Resource is propagated to the transformation as well (line 11). However, some other impacts can be co-evolved in many different ways. Indeed, in the first transformation rule, the elements *minTime* and *maxTime* can be deleted from the statements as shown in Listing 2 (line 5). But the whole assignment can be deleted as well. Imagine the property *name* from the PetriNet class is deleted, then it makes more sense to delete the complete assignment since the target of the assignment is *pn.name* (line 5). In case the whole class is deleted, then the whole transformation rule should be deleted. Here we already see that a delete change in the metamodel can be co-evolved in different manners to cover different possible co-evolutions.

Similar situation can even occur for the same impacting change. Take the moved property *isNormal* from Resource to Parameter. In the second transformation the navigation path to *isNormal* is extended with *parameter.first()* to access it from its first parameter (line 11). Whereas in the third transformation, for the same impacted *isNormal*, its navigation path is rather reduced from *parameter.resource.isNormal* to *parameter.isNormal* (lines 19 and 27). This emphasizes the fact that developers must be involved to decide which alternative co-evolutions should be applied.

An additional issue arises, if none of the proposed co-evolutions fits the developer intent. Then manual co-evolution is the only remaining option. However, a new co-evolution can still be proposed by composing the basic supported resolutions into a new one. It can be seen as analogous to how atomic changes are composed into complex changes. The next section presents our approach that addresses the above mentioned challenges by supporting a change propagation-based and a composition-based co-evolution of transformations

Listing 1: Excerpt of original transformation rules.

```
1  //Transform a Process to PetriNet
2  rule Process2PetriNet
3  transform p: simplepdl!Process
4  to pn: petrinet!PetriNet{
5  pn.name = p.name + "[" + p.minTime + "," + p.max-
       Time + "]";
6  }
7  // Transform a Resource to Place
8  rule Resource2PetriNet
9  transform resource: simplepdl!Resource
10 to p_resource: petrinet!Place{
11 p_resource.name = resource.name + "[" + resource.is-
       Normal + "]";
12 p_resource.tokensNB = resource.quantity;
13 p_resource.petriNet = resource.getProcess();
14 }
15 // Transform a Parameter to Arcs
16 rule Parameter2PetriNet
17 transform parameter: simplepdl!Parameter
18 to a_r2s: petrinet!Arc, a_f2r: petrinet!Arc{
19 if(parameter.resource.isNormal == true){
20 a_r2s.kind = '#normal';
21 } else {
22 a_r2s.kind = '#read_arc';
23 }
24 a_r2s.tokensNB = 1; a_r2s.source = p.resource;
25 a_r2s.target = thisModule.resolveTemp(p.
       workDefinition, 't_start');
26
27 if(parameter.resource.isNormal == true){
```

[1]The transformation is detailed in https://www.eclipse.org/atl/usecases/SimplePDL2Tina/
[2]https://www.eclipse.org/atl/
[3]https://www.eclipse.org/epsilon/doc/etl/

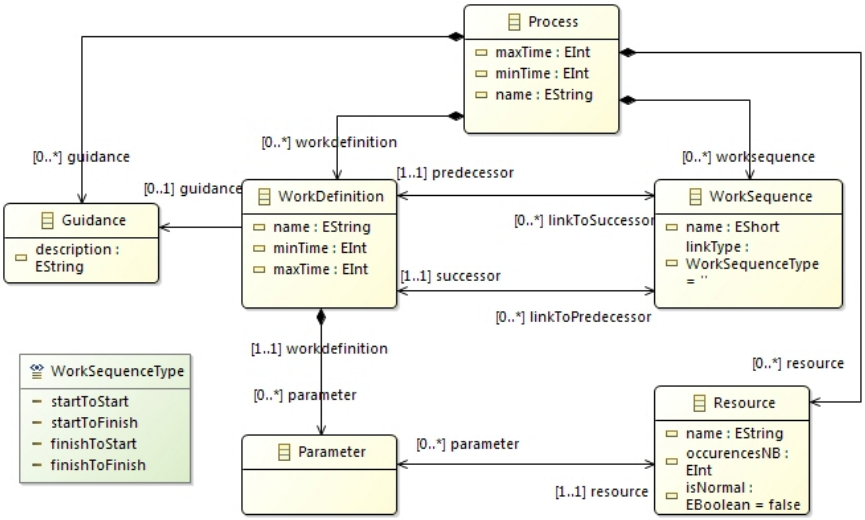

Figure 1: Excerpt of the SimplePDL metamodel.

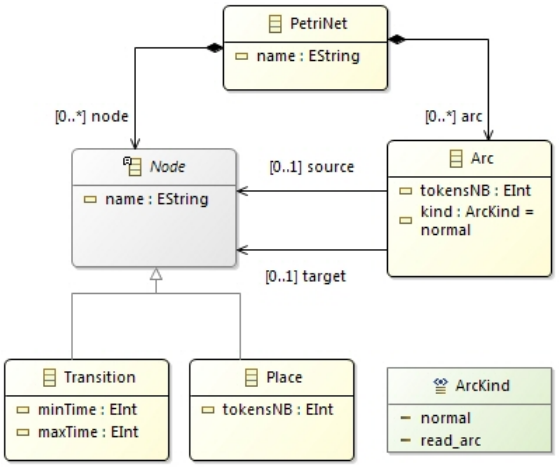

Figure 2: Excerpt of the PetriNet metamodel.

```
28 a_f2r.kind = '#normal';
29 } else {
30 a_f2r.kind = '#read_arc';
31 }
32 a_f2r.tokensNB = 1; a_f2r.target = p.resource;
33 a_f2r.source = thisModule.resolveTemp(p.
       workDefinition, 't_finish');
34 }
```

Listing 2: Excerpt of original transformation rules.

```
1 //Transform a Process to PetriNet
2 rule Process2PetriNet
3 transform p: simplepdl!Process
4 to pn: petrinet!PetriNet{
5 pn.name = p.name + "[" + "," + "]";
6 }
7 // Transform a Resource to Place
8 rule Resource2PetriNet
9 transform resource: simplepdl!Ressource
10 to p_resource: petrinet!Place{
11 p_resource.name = resource.URI + "[" + resource.pa-
       rameter.first().isNormal + "]";
12 p_resource.tokensNB = resource.quantity;
13 p_resource.petriNet = resource.getProcess();
14 }
15 // Transform a Parameter to Arcs
16 rule Parameter2PetriNet
17 transform parameter: simplepdl!Parameter
18 to a_r2s: petrinet!Arc, a_f2r: petrinet!Arc{
19 if(parameter.isNormal == true){
20 a_r2s.kind = '#normal';
21 } else {
22 a_r2s.kind = '#read_arc';
23 }
24 a_r2s.tokensNB = 1; a_r2s.source = p.resource;
25 a_r2s.target = thisModule.resolveTemp(p.
       workDefinition, 't_start');
26
27 if(parameter.isNormal == true){
28 a_f2r.kind = '#normal';
29 } else {
30 a_f2r.kind = '#read_arc';
31 }
32 a_f2r.tokensNB = 1; a_f2r.target = p.resource;
33 a_f2r.source = thisModule.resolveTemp(p.
       workDefinition, 't_finish');
34 }
```

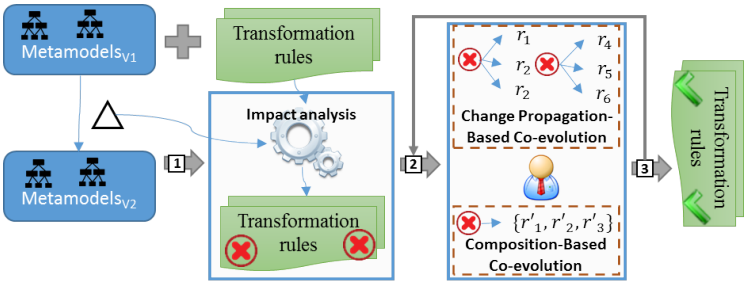

Figure 3: Overall co-evolution process of transformations

Table 1: Mapping table used for the impact analysis.

Metamodel Elements	Transformations	References to AST Nodes
$e_1[type, container]$	$trans_1, \ldots$	$ast_1, ast_2, \ldots$
...	...	...
$e_n[type, container]$	$trans_m, \ldots$	$ast_j, ast_j, \ldots$

3 OVERALL APPROACH

Figure 3 shows our overall co-evolution process of model transformations. It first identifies the metamodel changes to be propagated and then runs an impact analysis on the transformations 1. Then a change propagation-based co-evolution is applied 2. The developer is involved throughout the co-evolution and in case a co-evolution does not fit her intent, a composition-based co-evolution is provided as an alternative. Finally, the chosen resolutions are applied to co-evolve the transformations 3.

3.1 Change Detection and Impact Analysis

Before we start co-evolving transformations, we run an impact analysis to detect the transformations that must be co-evolved. In particular the impacted parts of the transformations as a single transformation may be impacted several times in different parts.

The first pre-requisite for the impact analysis is to identify the performed metamodel changes that can occur both in the source and target metamodels. The detection of the metamodel changes is performed with our previous work [26, 27]. It detects both *atomic* and *complex* changes [19]. *Atomic* changes are additions, removals, and updates of a metamodel element. *Complex* changes consist in a sequence of atomic changes combined together. For example, *push property* is a complex change where a property is moved from a superclass to its subclasses. This is composed of two atomic changes: delete property and add property.

With the detected metamodel changes, we run an impact analysis [23] to localize the impacted transformations. To this end, we parse the transformations to access their Abstract Syntax Tree (AST). We then establish a mapping between the metamodel elements $e_1, \ldots, e_n$ (with their types and containers) and the AST nodes using $e_1, \ldots, e_n$ in the transformations. Table 1 illustrates the content of our computed mapping table.

Algorithm 1 Transformations co-evolution

```
1:  function TransformationsCo-Evolution(Set<changes> changes, MappingTable MT, ResolutionCatalog RC)
2:      for all c ∈ changes do
3:          impacts ← MT.getAllCausedImpacts(c)
4:          for all i ∈ impacts do
5:              if changePropagationMode then
6:                  resolution ← ChangePropagation(i, c, RC)
7:              else                                  ▷ compositionMode
8:                  resolution ← ComposeResolution(RC)
9:              end if
10:             i.apply(resolution)
11:         end for
12:     end for
13: end function
```

We know from Kusel et al. [29] the changes that are impacting transformations (shown in column in Table 2), such as delete, update, and move changes. Hence, the impact analysis consists in accessing for each impacting metamodel change on a element e_i, the set of impacted transformations and their impacted AST nodes in Table 1. Note that during the co-evolution, Table 1 is also updated accordingly with the applied resolutions. For instance, when a rename element e occurs, it is also renamed in the table.

3.2 Transformation Co-evolution

Our work supports two modes of co-evolution: 1) a change propagation-based and 2) a composition-based co-evolution of transformations.

Algorithm 1 illustrates the overall co-evolution process. It first retrieves the impacted transformation parts by a given metamodel change. Then for each impact, developers can choose to either use our change propagation co-evolution (line 6) or our composition mechanism (line 8). Finally, the chosen resolution is applied on the impacted transformation part. These two possible co-evolution modes are further explained in the next two sections.

3.2.1 *Change Propagation-based Co-evolution.*

Algorithm 2 illustrates our change propagation-based co-evolution. For a given impact on a transformation by a given change, we first retrieve the appropriate resolutions that can propagate the impacting change.

Table 2 shows the resolutions that propagate each impacting change to the transformations. The resolutions are inspired from the co-evolution of other artifacts, such as models [14] or constraints [25], where they showed to be efficient and useful. In case alternative resolutions can be applied, we involve the developer in deciding which one fits her needs. This acts as a user acceptance of the resolution to be applied.

Note that we do not claim completeness of our catalog of resolutions. It rather represents possible propagations of the metamodel changes at the transformations level. To assess whether the catalog is sufficient and useful in practice, we will further evaluate to what extent our catalog of resolutions can correctly co-evolve impacted transformations w.r.t. the developers intent.

Table 2: Catalog of resolutions that propagates the metamodel changes.

Metamodel Changes	Resolutions
⋄ Delete element e	▷[$R0$] Remove e used as part of statement in the transformation rule (e.g., t.label = s.name + s.e → t.label = s.name) ▷[$R1$] Remove the statement using e in the transformation rule (i.e., a whole IF/loop/assignment statement, etc.) ▷[$R2$] Remove the whole transformation rule ▷[$R3$] Replace e with another one in the transformation rule
⋄ Rename element e	▷[$R4$] Rename e in the transformation rule
⋄ Generalize property p multiplicity from a single value to multiple values	▷[$R5$] Introduce a For loop statement to iterate on a collection ▷[$R6$] Introduce operation first() on a collection (e.g., $lng.p \rightarrow lng.first().p$) ▷[$R7$] Introduce operation last() on a collection (e.g., $lng.p \rightarrow lng.last().p$) ▷[$R8$] Introduce operation at(index) on a collection, where index is an expression given by the user (e.g., $lng.p \rightarrow lng.at(lng.size()-1).p.$ or $lng.at(0).p.$)
⋄ Move property p from class S to T	▷[$R9$] Extend navigation path of p (e.g., $lng.p \rightarrow lng.path.p$) ▷[$R10$] Reduce navigation path of p (e.g., $lng.path.p \rightarrow lng.p$)
⋄ Push property p from class Sup to Sub_1,...,Sub_n	▷[$R11$] Introduce a type test with an If (e.g., $t.name = s.p.name \rightarrow if(s.p.istypeof(Sub_1)\ or \ldots or\ s.p.istypeof(Sub_n))\{t.name = s.p.name\}$)
⋄ Extract class S to T with properties $p_1, \ldots, p_n$	▷[$R9$] Extend navigation path of p_i (e.g., $lng.p_i \rightarrow lng.path.p_i$) ▷[$R10$] Reduce navigation path of p (e.g., $lng.path.p_i \rightarrow lng.p_i$)
⋄ Inline class S to T with properties $p_1, \ldots, p_n$	▷[$R12$] Change the class type in the transformation rule from S to T (e.g., transform s:S to o:O {...} → Transform s:T to o:O {...}) ▷[$R2$] Remove the whole transformation rule ▷[$R10$] Reduce navigation path of p
⋄ Flatten hierarchy from class Sup to Sub_1,...,Sub_n with properties $p_1, \ldots, p_n$	▷[$R14$] Duplicate the transformation rule while changing the source or target class type from Sup to Sub_i ($i \in [1 \ldots n]$) ▷[$R2$] Remove the whole transformation rule

Algorithm 2 Change propagation-based co-evolution

```
1:  function ChangePropagation(Impact i, Change c, ResolutionCatalog RC)
2:      resolutions ← RC.getAppropriateResolutions(i, c)
3:      if resolutions.size > 1 then
4:          res ← UserDecision(resolutions)
5:      else
6:          if resolutions.size = 1 then
7:              res ← resolutions.first()
8:          end if
9:      end if
10:     return res
11: end function
```

3.2.2 *Composition-based Co-evolution.*

Our resolutions from Table 2 can be composed together, leading to a possible set of 2^n new resolutions ($n \in CatalgOfResolutions$). It would be overwhelming for developers to then choose the appropriate resolution to apply. Thus, we rather allow developers to compose their needed resolutions. This already showed to be useful in the context of models co-evolution by Herrmannsdoerfer et al. [17, 18, 20] who supported compositions of model resolutions.

Algorithm 3 illustrates our composition-based co-evolution, which has two main goals. First, it allows developers to freely select resolutions that would not have been proposed by our change propagation-based co-evolution Algorithm 2. For example, instead of propagating a change move property *id* by extending its path ([$R9$]). Developers may decide to deviate and apply another resolution that is not suggested to them, such as to delete *id* ([$R0$]), or to replace *id* with another element ([$R3$]).

Second, it also allows developers to compose resolutions together that would result in a new resolution that is not in our catalog in Table 2. For example, composing the resolutions replace an element with an add of the operation first() (e.g., $var.parent.id \rightarrow var.ancestors.id \rightarrow var.ancestors.first().id$). Another example would be composing the resolutions reduce path and then extend it with another path (e.g., $var.student.name \rightarrow var.name \rightarrow var.person.name$).

Note that not all possible compositions are valid. Hence, we define incompatible compositions that cannot be performed and we check them before to compose the developer's chosen resolutions (line 5). For example, an incompatible composition of resolutions is between delete resolutions and the rest of resolutions.

3.3 Prototype Implementation

Our approach's Java implementation handles Ecore/EMF metamodels and transformations from the Epsilon Transformation Language (ETL). It interfaces with our detection approach [24, 27] of metamodel changes to then run an impact analysis on the transformations. For each impacted part we propose resolutions. The resolutions are implemented with Java working on the ETL AST classes. In case of alternative resolutions, the developer can then choose the appropriate ones to be applied automatically.

Algorithm 3 Composition-based Co-evolution

```
1: function ComposeResolution(ResolutionCatalog RC)
2:     resolution ← ∅
3:     while ¬finished do
4:         cr ← RC.getChosenResolution()
5:         if isComposable(resolution, cr) then
6:             resolution ← resolution ∪ cr
7:         end if
8:     end while
9:     return resolution
10: end function
```

4 EVALUATION

This section presents the evaluation results of our co-evolution approach. We first present the dataset and evaluation process. Then, we present the research questions and discuss the obtained results. Time performance of the co-evolution is measured as well.

4.1 Data Set

This section presents the used data set in our evaluation. Our case studies are divided in two groups. The first group is used to evaluate the change propagation-based co-evolution and the second is used to evaluate our composition-based co-evolution.

The first group consists of 14 case studies where each case study contains: *1)* original and evolved source/target metamodels, *2)* original and evolved ETL transformations. All evolutions in our case studies of the metamodels and ETL transformations have been performed manually by developers. We thus consider the manual transformations' co-evolution as the reference (i.e., ground truth) to which we compare our automatic co-evolution.

The source metamodels cover different domains of competitions (e.g., sport, e-game, etc.) and the target metamodels cover a betting domain, where bets are put on games from the source competitions. These case studies come from our last year Master's MDE lecture performed over 4 months. Those 14 case studies are from students who had a past developer experience or who were working as part time developers (making an average of 1 year experience). Students built their own modeling languages and used different modeling technologies (Sirius and GMF[4]). Evolution and co-evolution were part of the maintenance phase of the modeling languages.

The second group consists of three case studies taken from the catalog of ETL Epsilon[5]. The three case studies transform a tree model to a graph model, an object oriented model to a data base model, and a flowchart model to an HTML model.

Table 3 gives the selected case studies and the number of the transformation rules. Table 4 further gives details on the size of the metamodels and the transformations, both for original and evolved versions. Note that in the first group of case studies, only the source metamodels evolved, whereas in the second group the target metamodels evolved.

The data set is archived in the FigShare platform[6] to be used for reproducibility and comparison purposes.

[4]https://www.eclipse.org/sirius/ and http://www.eclipse.org/modeling/gmp/
[5]git://git.eclipse.org/gitroot/epsilon/org.eclipse.epsilon.git
[6]https://figshare.com/s/13230c5cef89d854a295

Table 3: Co-evolution case studies.

	Case studies	N^o of original rules+helpers	N^o of evolved rules+helpers
Group 1	CS1:PokerTournament	4	4
	CS2:DanceTournament	4	4
	CS3: eSportsLeague	4	4
	CS4:SpaceRace	4	4
	CS5:SoccerEmModel	5	5
	CS6:languageolympics	5	5
	CS7:BasketballLeague	9	9
	CS8:Hackathon	4	4
	CS9:soccer_Tournament	4	4
	CS10:codingcontest	4	4
	CS11:nfl	12	12
	CS12:basketball	3	3
	CS13:golf_competition	8	8
	CS14:MPRPS	6	6
Group 2	CS15:tree2graph	1	1
	CS16:oo2db	9	9
	CS17:flowchart2html	19	19

4.2 Evaluation Process

To run this experiment, we first evaluate our change propagation-based co-evolution on the first group of case studies. Here, we measure the accuracy of our co-evolution tool by comparing for the same set of transformations how they were manually co-evolved by developers against how they are automatically co-evolved by our tool. This allows us to measure the correctness percentage reached by our co-evolution approach. The correctness metric of our co-evolution is computed as follows:

$$correctness = \frac{ProposedResolutions \cap ExpectedResolutions}{ExpectedResolutions}$$

The *ProposedResolutions* are the resolutions applied by our approach (from Table 2) and the *ExpectedResolutions* are the actual manually performed resolutions by developers. The correctness varies from 0 to 1, i.e., 0% to 100%.

After that we evaluate our composition-based co-evolution by asking developers to apply it on the second group of case studies.

Herein we do not have versioned metamodels and ETL transformations. Thus, we evolved the metamodels by applying impacting changes from Table 2. We asked eight developers to use our composition-based co-evolution on five impacted transformations, one for `CS15:tree2graph`, and two for both `CS16:oo2db` `CS17:flowchart2html`. We then analyzed the applied 40 resolutions. Finally, note that the metamodel changes from version *1* to version *2* are known for both groups. Based on those changes we run an impact analysis to identify the impacted transformations that will be co-evolved.

4.3 Research Questions

This section defines three research questions (RQ) to assess in particular the applicability, correctness, and the usefulness of our work. The three research questions are as follows:

Table 4: Detailed co-evolution case studies of the used metamodels and ETL transformations.

	Case Study /Domains	Size of source Metamodels in N^o of elements		Size of target Metamodels in N^o of elements		Size of Transformations in N^o of LOC	
		Original	Evolved	Original	Evolved	Original	Evolved
Group 1	CS1:PokerTournament	89	85	219	n/a	143	143
	CS2:DanceTournament	86	84	219	n/a	46	46
	CS3: eSportsLeague	57	52	219	n/a	54	54
	CS4:SpaceRace	55	59	219	n/a	47	49
	CS5:SoccerEmModel	73	76	219	n/a	77	77
	CS6:languageolympics	84	83	219	n/a	97	97
	CS7:BasketballLeague	74	78	219	n/a	85	85
	CS8:Hackathon	57	55	219	n/a	27	27
	CS9:soccer_Tournament	63	54	219	n/a	57	57
	CS10:codingcontest	119	108	219	n/a	54	57
	CS11:nfl	109	113	219	n/a	132	129
	CS12:basketball	75	73	219	n/a	56	56
	CS13:golf_competition	49	49	219	n/a	94	94
	CS14:MPRPS	56	66	219	n/a	100	100
Group 2	CS15:tree2graph	7	n/a	15	20	15	n/a
	CS16:oo2db	67	n/a	30	34	142	n/a
	CS17:flowchart2html	28	n/a	321	333	169	n/a

RQ1: To what extend and how fast can our change propagation-based co-evolution handle the impacted Transformations? If we cannot propose a co-evolution for all impacted transformations, we can conclude that our change propagation-based co-evolution is insufficient. This assesses the overall applicability of our change-propagation.

RQ2: What is the correctness percentage of our change propagation-based co-evolution? This measures the correctness level of our co-evolution approach, i.e., to which extent it proposes the expected resolutions. In so doing, we also assess its usefulness.

RQ3: To what extent can our composition-based co-evolution be useful to developers in customizing their co-evolution? Here we aim to assess its usability and usefulness when used as an alternative to our change propagation-based co-evolution.

4.4 Results

We now discuss the results for our research questions.

4.4.1 RQ1.

With our impact analysis we were able to find all impacted transformations' parts for which resolutions were proposed by our change propagation-based co-evolution. All impacted parts were co-evolved and a total of 83 resolutions were applied during the change propagation. This shows the applicability of our change propagation-based co-evolution that was able to handle all impacted transformations. Table 5 shows the applied resolutions for each case study during co-evolution. In total, the 83 applied resolutions were covered by 8 resolution types from our catalog in Table 2, i.e., [$R0$], [$R1$], [$R3$], [$R4$], [$R5$], [$R6$], [$R9$], [$R10$].

The detection of an impacted transformation part and the application of its resolution was a matter of milliseconds (ms) with an average of 3ms. The evaluation was run on a Windows 7 PC with a Core i7 3.4GHz and 16GB RAM.

4.4.2 RQ2.

During co-evolution, when alternative resolutions were possible to apply, we selected the one that is *as close as possible* to the expected resolution. Hence, after that we could measure the correctness of our co-evolution.

Figure 4 shows the measurement of the *correctness* metric on our 14 case studies. On average, our change propagation-based co-evolution was able to reach 96% correctness. This means that our applied resolutions propagating the impacting metamodel changes were the ones expected by the developers, in 96% of the cases. This shows the usefulness of using change propagation-based co-evolution while meeting the developer's co-evolution needs.

In 12 out of the 14 case studies from the first group, a 100% correctness was reached. In the `CS7:BasketballLeague` case study we reached an 83% correctness (10/12). Due to a delete of the property *name* in the class `PlayOffs`, the developer's manual co-evolution was different from our proposed resolutions ([$R0$], [$R1$], [$R2$], [$R3$]). Instead, the property *name* and its navigation path, used in the two below ETL statements, was replaced entirely. The first *p.name* was replaced by a call to a helper operation *getGroupIndex(p)*. Whereas, the second *p.name* was replaced to another element with another navigation path *gr.title*, as shown below.

gr.title = p.name; $\Longrightarrow$ *gr.title = 'Playoffs' + getGroupIndex(p);*

gr.description = '...' + p.name; $\Longrightarrow$ *gr.description = '... ' + gr.title;*

In the `CS10:codingcontest` case study, we reached a 57% correctness (4/7). Herein, in response to the move property *localDate*

Table 5: Number of applied resolutions in our change propagation-based co-evolution for each case study.

	Case studies	Applied resolutions	N^o of application
Group 1	CS1:PokerTournament	▷[R0]	2
	CS2:DanceTournament	▷[R4]	6
		▷[R10]	1
	CS3: eSportsLeague	▷[R4]	2
	CS4:SpaceRace	▷[R5]	1
	CS5:SoccerEmModel	▷[R4]	8
		▷[R9]	7
	CS6:languageolympics	▷[R4]	1
		▷[R6]	1
	CS7:BasketballLeague	▷[R3]	2
		▷[R4]	3
		▷[R9]	7
	CS8:Hackathon	▷[R4]	2
	CS9:soccer_Tournament	▷[R4]	7
	CS10:codingcontest	▷[R4]	2
		▷[R6]	1
		▷[R9]	3
		▷[R10]	1
	CS11:nfl	▷[R1]	1
		▷[R3]	1
		▷[R4]	4
	CS12:basketball	▷[R4]	2
	CS13:golf_competition	▷[R4]	3
		▷[R6]	3
		▷[R10]	1
	CS14:MPRPS	▷[R4]	4
		▷[R6]	3
		▷[R9]	4

from class `Contest` to class `Location`. The three below ETL statements were impacted for which we extended their navigation path with the resolution [R9]. However, the developer simply decided to delete (∅) them with the resolution [R1] as shown below.

grp.startDate = cc.localDate; ⟹ ∅

grp.endDate = cc.localDate; ⟹ ∅

online.date = cc.localDate; ⟹ ∅

4.4.3 RQ3.

The composition-based co-evolution targets two main goals. The first one is to allow developers to freely select a resolution that meets their needs, regardless the impacting metamodel changes. For example, in the case of `CS10:codingcontest` case study, the developer could have directly chosen the resolution [R1] instead of the proposed [R9] and [R10]. Thus, improving the co-evolution correctness from 57% to 100% in this case.

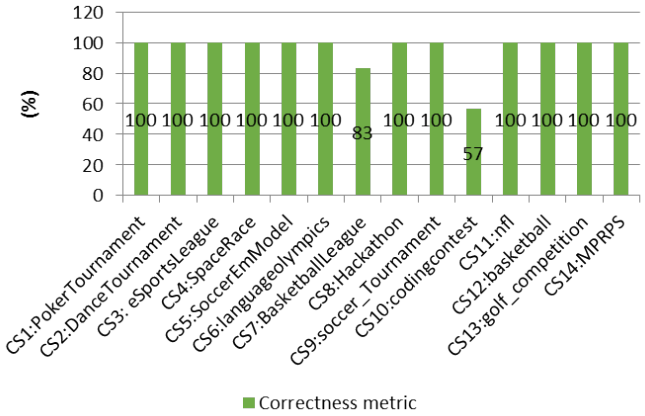

Figure 4: Correctness measurement for our change propagation-based co-evolution.

The second goal is to allow developers to compose new resolutions together out of our catalog in Table 2. To evaluate our composition-based co-evolution, we used the second group of case studies and asked eight developers (working in our Computer Science Department at the Johannes Kepler University) to use it when co-evolving the transformations. Their professional programming experience, was an average of 2 years and 1 month as developers (with minimum of half year to a maximum of 5 years). All participants were given the metamodel evolution, the five impacted transformations (with highlighted impacted parts), and they were asked to use our catalog of resolutions to co-evolve the transformations. We did not make it mandatory to compose resolutions to avoid bias in the resulting resolutions. We only asked them to select resolutions for each transformation. Hence, we avoid irrelevant composed resolutions that would have been proposed just for the sake of composition. Before we started the experiment, we took 15 minutes to explain the three case studies so that the participants get an understanding of the transformations. Below we report on the observed results.

In the `CS15:tree2graph` the applied resolutions were: $[R9] \times 7$ (i.e., applied seven times), $\{[R9] \cup [R8]\} \times 1$. In the `CS16:oo2db` the applied resolutions were: $[R1] \times 1$, $[R11] \times 8$, $[R12] \times 2$, $[R14] \times 2$, $\{[R14] \cup [R1]\} \times 2$, $\{[R14] \cup [R11]\} \times 1$. Finally, in the `CS17:flowchart2html` the applied resolutions were: $[R1] \times 1$, $[R2] \times 2$, $[R5] \times 1$, $[R9] \times 9$, $[R14] \times 1$, $\{[R9] \cup [R3]\} \times 2$.

Most of the applied resolutions (34/40) were not the result of compositions. Among those, 18 resolutions were applied by all developers propagating the impacting metamodel changes. For example, due to a push property in `CS16:oo2db`, the resolution [R11] was applied eight times by five developers. However, seven developers applied other 15 resolutions but not from the ones propagating the impacting metamodel changes. For example, due to same push property in `CS16:oo2db`, the resolution [R12] was applied twice by one developer who changed the target super class to the subclass where the property is pushed. This already shows that our composition mechanism is a useful complement to our change propagation mechanism. Moreover, four developers chose to compose six resolutions in order to co-evolve six impacted transformations. The applied composed resolutions are: 1) $\{[R9] \cup [R3]\}$, 2) $\{[R9] \cup [R8]\}$,

3) {[$R14$] ∪ [$R1$]}, and 4) {[$R14$] ∪ [$R11$]}. The first and the second compositions were each applied two times. These results show that our composition-based co-evolution is also useful for developers to compose resolutions in particular cases. This is further highlighted in a post questionnaire where seven participants judged our composition mechanism as useful and one judged it as very useful[7].

4.5 Discussion and Limitations

Our change propagation-based co-evolution was stress tested on 14 case studies. It showed to be efficient and useful in co-evolving model transformations with an average correctness of 96%. The main advantage is to not overwhelm developers with a large set of possible resolutions per impacted transformation. This is due to the concept of change propagation that leverages on the metamodel changes to reflect them on the transformations.

The main drawback is that the change propagation-based co-evolution is limited to our catalog in Table 2. However, in our evaluation it showed to be sufficient to deal correctly with most cases of co-evolution (96%). Nonetheless, to address this limitation, we proposed a composition-based co-evolution to deal in particular with cases where transformations are co-evolved in different and unexpected way. In our experiment with eight developers, half composed resolutions at least once to co-evolve their transformations. Hence showing the usefulness of supporting a composition-based co-evolution.

Moreover, we also observed two cases of refactorings that occurred in addition to the expected co-evolutions. For example, the multiplicity of the property *organizer* was increased from 1 to *. Thus, we proposed resolution [$R6$] that introduces the operation `first()`. This was indeed the expected resolution. However, an If statement was also introduced to distinguish the cases of single value and multiple values in the property *organizer*, as shown below. This paper focused on the co-evolution of impacted transformations and refactoring was out of the scope.

```
grp.description = "Organized by " + cc.organizer.name;
⟹
if(cc.organizer.size() > 1) {
    grp.description = "Jointly organized";
} else {
    grp.description = "Organized by " + cc.organizer.first().name;
}
```

5 THREATS TO VALIDITY

This section discusses internal, external, and conclusion threats to validity after Wohlin et al. [42] w.r.t. our three research questions.

5.1 Internal Validity

The first group of used case studies comes from our last year master students. However, recent studies [35, 37] have shown that students are valid subjects for experiments and that students are well representative when it comes to new developing tasks. This was the case by building modeling languages and maintaining them. Nonetheless, to further reduce this threat we selected case studies from master students that already had a working experience and that were working in parallel in a half time programming job. Thus, we aimed at selecting participants near to a junior experienced developer with an average of one year experience. In the second group of case studies we performed the metamodel evolution to cause impacts on transformations. To reduce the risk of applying non-realistic changes, we performed changes that are similar to those applied by developers in the well-know metamodel evolutions, such as those in UML[33] or GMF[12].

Moreover, our participants in our experiment are from our Computer Science Department. To mitigate the threat of results from our participants being in our favor. We only asked them to select resolutions to co-evolve transformations without informing them about the goal of the experiment. We also selected participants who are not working in the field of model transformations and metamodel evolution/co-evolution. They were experts in software product lines, traceability, collaboration, and cyber physical production systems, yet, with modeling knowledge.

5.2 External Validity

We implemented and evaluated our approach for EMF/Ecore and ETL. Other languages, such as ATL, use different syntax but conceptually use the same constructions as in ETL (e.g., transformation rules, from source to target elements, etc.) Although we are confident that the co-evolution would be applicable for other transformation languages, we cannot generalize our results to all transformation languages. Further experimentation on other languages is necessary. However, the only requirement to apply our approach to other languages is to have access to the ASTs of the parsed transformations and to adapt our resolutions to the new ASTs' structure.

5.3 Conclusion Validity

Our evaluation gave promising results, showing that our change propagation-based co-evolution is fast and useful with an average of 96% of correctness. The evaluation results also show that our composition-based co-evolution showed to be a useful in supporting compositions of new resolutions that are not in our catalog in Table 2. However, we only evaluated it on three case studies with eight developers. To have more insights and evidence, further evaluation is needed on more case studies with more participants.

6 RELATED WORK

The main idea of change propagation was already proposed and used for different purposes, such as by Hassan et al. [13] who used change propagation in the context of source code, Cubranic et al. [5] who used it to suggest relevant software development artifacts, or Demuth el al. [6] who used it for models co-evolution. In our work we use change propagation to co-evolve transformations.

Many approaches addressed the co-evolution of models [15, 34]. As transformations are different from models, the existing techniques [15, 34] cannot necessarily be used for transformations. Indeed, as transformations refer to both the source and target metamodels. A change in either one can have heavier consequences on transformations, e.g., leading to creation of new transformation rules. However, few approaches only focused on Transformations'

[7] Between 'useless – little useful – neutral – useful – very useful'.

co-evolution. In this section, we present the main approaches that co-evolve transformations and we compare them w.r.t. our work.

Alkhazi et al. [1] proposed an automatic refactoring approach of ATL transformations . They supported 10 types of refactorings that are different from our resolutions in Table 2, which is expected since the purpose of co-evolution and refactoring is different. However, other approaches focused on the co-evolution of transformations.

Vieira et al. [39, 40] proposed an impact analysis approach of ATL transformations, but without proposing any co-evolution. Other approaches proposed to co-evolve transformations [1, 9–11, 29, 30]. All those approaches include an impact analysis to identify the impacted transformations and do not delegate it to the users, which is similarly performed in our approach as well.

Mendez et al. [30] was the first attempt to handle this issue. They proposed a set of resolutions to be applied on the impacted transformations, similarly as in our approach. Ruscio et al. [8, 9] proposed an approach where in addition to the impact analysis, they also evaluate the co-evolution cost of the transformations. If the cost is not expensive, then an automatic co-evolution is performed with EMFMigrate. Ruscio et al. [7] further explored the variability of the co-evolution due to the possible alternative resolutions. Garcia et al. [11] presented a change propagation-based co-evolution of ATL transformations. They implemented their co-evolution by means of ATL transformations, where each transformation is a resolution. Garces et al. [10] proposed an approach that computes a chain of adaptations to co-evolve ATL transformations based on the metamodel changes. They further explored different orders of the chain of adaptations, in particular to filter incorrect orders. Garcia et al. [11] and Garces et al. [10] also proposed their own approach for change detection in contrast to other existing approaches. Kusel, et al. [29] proposed an approach that performs transformations co-evolution based on impacting atomic changes. They further proposed a composition mechanism from atomic changes $a_1, ..., a_n$ to complex changes, which triggers the composition of the resolutions associated with the atomic changes.

To the best of our knowledge, apart from Ruscio et al. [7], all existing approaches support unique resolution per impacted transformation part. They do not offer alternatives to developers. We also investigated their catalog of resolutions. We found that the supported resolutions are different from an approach to another with some resolutions that are common to all approaches (e.g., [$R9$]). We further found that no existing approach supports all the resolutions that we were able to apply on our case studies, as shown in Table 6. This means that the existing approaches cannot fully automate the performed co-evolution in our case studies. For example, they would have reached a lower correctness co-evolution % than our reached 96% in the first group of case studies.

Furthermore, except from Ruscio et al. [8, 9], none allows to deviate from the proposed co-evolution and to choose a different one. Only Ruscio et al. [8, 9] allow developers to manually replace or refine the supported resolutions. However, none allows to compose their own resolutions. The composition mechanism proposed by Kusel, et al. [29] support specific compositions, i.e., resolutions from atomic changes to a complex change. In contrast, our composition mechanism allows all possible combinations of resolutions that are not in conflict with each other. Our composition-based co-evolution is similar to the one proposed by Herrmannsdoerfer et al. [17, 18, 20].

Table 6: Supported resolutions used in our case studies by the existing approaches for transformations' co-evolution.

Resolutions	Kusel, et al. [29]	Garcia et al. [11]	Ruscio et al. [8, 9]	Garces et al. [10]	Mendez et al. [30]	Our work
[$R0$]	×	✓	×	×	×	✓
[$R1$]	✓	✓	×	✓	✓	✓
[$R2$]	✓	✓	✓	✓	✓	✓
[$R3$]	×	×	✓	×	✓	✓
[$R4$]	✓	×	✓	✓	✓	✓
[$R5$]	×	✓	×	×	×	✓
[$R6$]	×	✓	×	×	×	✓
[$R9$]	✓	✓	✓	✓	✓	✓
[$R10$]	×	×	×	✓	×	✓
[$R11$]	×	×	×	×	×	✓
[$R12$]	✓	×	✓	×	×	✓
[$R14$]	×	✓	×	×	×	✓

They allowed developers to compose existing resolutions into a new one. However, they proposed it for the co-evolution of model instances and not transformations as in our work. Resolutions for transformation are different from those for model instances. To the best of our knowledge, no existing approach could have supported the composed resolutions ($\{[R9] \cup [R3]\}$, $\{[R9] \cup [R8]\}$, $\{[R14] \cup [R1]\}$, and $\{[R14] \cup [R11]\}$) that were applied by four of our eight developers.

7 CONCLUSION

We presented a co-evolution approach of model transformations when metamodels evolve. It supports a change propagation-based co-evolution and a competition-based co-evolution. The former leverages on the metamodel changes and propagate them on the transformation. The latter allows developers to freely select which revolutions to apply and further to compose resolutions together.

Our change propagation-based co-evolution was evaluated on 14 case studies. It showed to be useful by reaching a correct co-evolution of 96%. The evaluation of our competition-based co-evolution with eight developers showed to be useful. It allowed developers to apply resolutions that do not necessary propagate the impacting change. But more importantly it allowed half of the developers to compose resolutions during co-evolution.

As future work, we plan to explore further the composition of resolutions and investigate what is the minimum set of resolutions that can be sufficient for a wide range composition. We further plan to use meta-heuristic algorithms to search for possible useful compositions of resolutions. Thus, we could directly suggest them to developers. Finally, we will evaluate our approach with more participants to investigate the frequency of composed resolutions. Then we could even add the more frequent compositions to our catalog of Table 2.

Acknowledgment. We would like to thank Horacio Hoyos Rodriguez for his feedback and help with ETL implementation. The

research leading to these results has received funding from the Austrian Science Fund (FWF) under the grants P25289-N15.

REFERENCES

[1] Bader Alkhazi, Terry Ruas, Marouane Kessentini, Manuel Wimmer, and William I Grosky. 2016. Automated refactoring of ATL model transformations: a search-based approach. In *The ACM/IEEE 19th MODELS*. 295–304.

[2] Benoît Combemale, Pierre-Loïc Garoche, Xavier Crégut, Xavier Thirioux, and François Vernadat. 2007. Towards a Formal Verification of Process Model's Properties-SimplePDL and TOCL Case Study. In *9th International Conference on Enterprise Information Systems*. 80–89.

[3] Xavier Crégut, Benoit Combemale, Marc Pantel, Raphaël Faudoux, and Jonatas Pavei. 2010. Generative technologies for model animation in the TopCased platform. In *European Conference on Modelling Foundations and Applications*. Springer, 90–103.

[4] Jesús Sánchez Cuadrado, Esther Guerra, and Juan de Lara. 2016. Quick fixing ATL transformations with speculative analysis. *Software & Systems Modeling* (2016), 1–35.

[5] Davor Čubranić and Gail C Murphy. 2003. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th international Conference on Software Engineering*. IEEE Computer Society, 408–418.

[6] Andreas Demuth, Markus Riedl-Ehrenleitner, Roberto E Lopez-Herrejon, and Alexander Egyed. 2016. Co-evolution of metamodels and models through consistent change propagation. *Journal of Systems and Software* 111 (2016), 281–297.

[7] Davide Di Ruscio, Juergen Etzlstorfer, Ludovico Iovino, Alfonso Pierantonio, and Wieland Schwinger. 2017. A feature-based approach for variability exploration and resolution in model transformation migration. In *European Conference on Modelling Foundations and Applications*. Springer, 71–89.

[8] Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2011. What is needed for managing co-evolution in mde?. In *Proceedings of the 2nd International Workshop on Model Comparison in Practice*. ACM, 30–38.

[9] Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2013. A methodological approach for the coupled evolution of metamodels and atl transformations. In *ICMT*. Springer, 60–75.

[10] Kelly Garcés, Juan M Vara, Frédéric Jouault, and Esperanza Marcos. 2014. Adapting transformations to metamodel changes via external transformation composition. *Software & Systems Modeling* 13, 2 (2014), 789–806.

[11] Jokin García, Oscar Diaz, and Maider Azanza. 2013. Model transformation co-evolution: A semi-automatic approach. *SLE* 7745 (2013), 144–163.

[12] GMP. 2015. Graphical Modeling Project. Graphical Modeling Framework (GMF). http://www.eclipse.org/modeling/gmp/. (2015).

[13] Ahmed E Hassan and Richard C Holt. 2004. Predicting change propagation in software systems. In *Software maintenance, 2004. proceedings. 20th ieee international conference on*. IEEE, 284–293.

[14] Regina Hebig, Djamel Eddine Khelladi, and Reda Bendraou. 2015. Surveying the corpus of model resolution strategies for metamodel evolution. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 135–142.

[15] Regina Hebig, Djamel Eddine Khelladi, and Reda Bendraou. 2017. Approaches to co-evolution of metamodels and models: A survey. *IEEE Transactions on Software Engineering* 43, 5 (2017), 396–414.

[16] Zef Hemel, Lennart CL Kats, Danny M Groenewegen, and Eelco Visser. 2010. Code generation by model transformation: a case study in transformation modularity. *Software and Systems Modeling* 9, 3 (2010), 375–402.

[17] Markus Herrmannsdoerfer. 2011. COPE–A Workbench for the coupled evolution of metamodels and models. In *Software Language Engineering*. Springer, 286–295.

[18] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. 2009. COPE-automating coupled evolution of metamodels and models. In *ECOOP 2009–Object-Oriented Programming*. Springer, 52–76.

[19] Markus Herrmannsdoerfer, Sander D. Vermolen, and Guido Wachsmuth. 2011. An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. In *Software Language Engineering*, Malloy, Staab, and Brand (Eds.). 163–182.

[20] Markus Herrmannsdörfer and Guido Wachsmuth. 2014. Coupled evolution of software metamodels and models. In *Evolving Software Systems*. Springer, 33–63.

[21] John Hutchinson, Mark Rouncefield, and Jon Whittle. 2011. Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 633–642.

[22] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. 2011. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 471–480.

[23] Ludovico Iovino, Alfonso Pierantonio, and Ivano Malavolta. 2012. On the Impact Significance of Metamodel Evolution in MDE. *Journal of Object Technology* 11, 3 (2012), 3–1.

[24] Djamel Eddine Khelladi, Reda Bendraou, and Marie-Pierre Gervais. 2016. Adroom: a tool for automatic detection of refactorings in object-oriented models. In *ICSE Companion*. ACM, 617–620.

[25] Djamel Eddine Khelladi, Reda Bendraou, Regina Hebig, and Marie-Pierre Gervais. 2017. A semi-automatic maintenance and co-evolution of OCL constraints with (meta) model evolution. *Journal of Systems and Software* 134 (2017), 242–260.

[26] Djamel Eddine Khelladi, Regina Hebig, Reda Bendraou, Jacques Robin, and Marie-Pierre Gervais. 2015. Detecting complex changes during metamodel evolution. In *CAISE*. Springer, 263–278.

[27] Djamel Eddine Khelladi, Regina Hebig, Reda Bendraou, Jacques Robin, and Marie-Pierre Gervais. 2016. Detecting complex changes and refactorings during (meta) model evolution. *Information Systems* (2016).

[28] Djamel Eddine Khelladi, Horacio Hoyos Rodriguez, Roland Kretschmer, and Alexander Egyed. 2017. An Exploratory Experiment on Metamodel-Transformation Co-Evolution. In *Asia-Pacific Software Engineering Conference (APSEC), 2017 24th*. IEEE, 576–581.

[29] Angelika Kusel, Jurgen Etzlstorfer, Elisabeth Kapsammer, Werner Retschitzegger, Wieland Schwinger, and Johannes Schonbock. 2015. Consistent co-evolution of models and transformations. In *ACM/IEEE 18th MODELS*. 116–125.

[30] David Mendez, Anne Etien, Alexis Muller, and Rubby Casallas. 2010. Towards transformation migration after metamodel evolution. *ME Wokshop@MODELS* (2010).

[31] Tom Mens, Gabriele Taentzer, and Olga Runge. 2007. Analysing refactoring dependencies using graph transformation. *Software and Systems Modeling* 6, 3 (2007), 269.

[32] Tom Mens and Pieter Van Gorp. 2006. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science* 152 (2006), 125–142.

[33] OMG. 2015. Object Management Group. Unified Modeling Language (UML). http://www.omg.org/spec/UML/. (2015).

[34] Richard F Paige, Nicholas Matragkas, and Louis M Rose. 2016. Evolving models in Model-Driven Engineering: State-of-the-art and future challenges. *JSS* 111 (2016), 272–280.

[35] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. 2015. Are students representatives of professionals in software engineering experiments?. In *ICSE-Volume 1*. IEEE Press, 666–676.

[36] Shane Sendall and Wojtek Kozaczynski. 2003. Model transformation: The heart and soul of model-driven software development. *IEEE software* 20, 5 (2003), 42–45.

[37] Mikael Svahnberg, Aybüke Aurum, and Claes Wohlin. 2008. Using students as subjects-an empirical evaluation. In *2nd ESEM*. ACM, 288–290.

[38] Juha-Pekka Tolvanen and Steven Kelly. 2009. MetaEdit+: defining and using integrated domain-specific modeling languages. In *The 24th ACM SIGPLAN conference companion on OOPSLA*. 819–820.

[39] Andreza Vieira and Franklin Ramalho. 2014. Metrics to measure the change impact in ATL model transformations. In *International Conference on Product-Focused Software Process Improvement*. Springer, 254–268.

[40] Andreza Vieira and Franklin Ramalho. 2016. Towards Measuring the Change Impact in ATL Model Transformations. *International Journal of Software Engineering and Knowledge Engineering* 26, 02 (2016), 153–181.

[41] Guido Wachsmuth. 2007. Metamodel adaptation and model co-adaptation. In *ECOOP*. Springer, 600–624.

[42] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.