An Exploratory Experiment on Metamodel-Transformation Co-Evolution

Djamel Eddine Khelladi, Horacio Hoyos Rodriguez, Roland Kretschmer, and Alexander Egyed
Institute for Software Systems Engineering
Johannes Kepler University Linz, Austria
Email: {djamel_eddine.khelladi, horacio.hoyos_rodriguez, roland.kretschmer, alexander.egyed}@jku.at

***Abstract*—Metamodels, like any other software artifacts evolve throughout time. As a consequence, all dependent artifacts may need to be co-evolved accordingly, including model transformations. Transformations are a key component of an automated development solution, thus it is crucial to automate their co-evolution while guaranteeing that they remain correct. However, there is little known about what aspects and characteristics must be automated in a manual co-evolution and in particular how it should be correctly automated. Few approaches exist, but it is not clear to what extent those approaches are able to automate the manual co-evolution of model transformations.**

In this paper, we report on an exploratory experiment we conducted to better understand the co-evolution of transformations in practice and to assess the usefulness of the current existing techniques. 15 participants were involved in our experiment to monitor how they co-evolve transformation rules in response to metamodel evolution. Our analysis results show that while existing approaches support the user with an automatic impact analysis, they do not consider proposing a very large spectrum of alternative resolutions. Among the 14 resolutions that occurred in our experiment, on average only 4 (up to 6) were supported by the existing approaches.

I. Introduction

Model-Driven Engineering (MDE) has shown to be effective in the development and maintenance of large scale and embedded systems [9]. Today modeling languages play a significant role in all phases of development processes [25].

The very foundation of modeling languages are their metamodels [9]. 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 writing model transformations. Model transformations play a crucial role in *MDE* [23], [17] by specifying and automating the transformation of a source model to a target model (i.e., model-to-model transformation) or to a target text (i.e., model-to-text transformation).

One of the foremost challenges to deal with in *MDE* is evolution of metamodels and its impact on artifacts that use the metamodels as a foundation. As a result of metamodel evolution, transformations may become invalid and hence may need to be co-evolved accordingly. Knowing that model transformations are used for different activities such as code generation [8], model refactoring [16] and migration [26], it is crucial to understand how to efficiently co-evolve the transformations whenever the metamodel evolves.

Automating the co-evolution is widely agreed on [6], however, little is known about the requirements of such an automation task [7]. In particular, does providing impact analysis to the developers help them, or is it already covered by the transformation languages/editors? Should automatic approaches provide unique resolutions per impacted transformation? Or should an approach cover all possible alternative resolutions? To what extent does the knowledge of metamodel evolution changes help in co-evolving the transformations?

The topic of transformation co-evolution has been increasingly investigated in the last years. However, little is known whether the proposed approaches are currently sufficient to automate various scenarios of transformations' co-evolution. While existing studies compared transformation tools and transformation-based co-evolution [10], [20], [21], [19], to the best of our knowledge no study has investigated the manual co-evolution of transformations and to what extent it could be automated with the existing approaches.

In this paper, we conduct an experiment with 15 participants that manually co-evolve model-to-model and model-to-text transformations. We investigate the above questions and provide new insights on aspects and characteristics of an efficient automated co-evolution, while highlighting the main learned lessons. We further evaluate the capabilities of the current existing approaches in terms of supported resolutions, i.e., to what extent existing approaches can be used to automate the manually performed resolutions in our experiment. Finally, we also reflect on how the state of the art can be improved in future work.

The experiment results show that transformation languages provide no support for impact analysis and errors are only reported during execution. Existing co-evolution approaches were able to support on average 4 resolutions (up to 6) out of the 14 that have been manually applied by our participants. This is partially explained due to the fact that alternative resolutions were not considered by the existing approaches.

This paper is further structured as the following. Section II describes the method of our experiment, the selected participants, the used data and research questions. The results' analysis is then given in Section III. Section IV discusses a follow-up survey with the participants. Sections V and VI discuss respectively the threats to validity and related works before Section VII concludes the paper.

II. Research Methodology

Our experiment involved 15 participants in the context of an *MDE* lecture. The participants had to evolve the metamodel and then to co-evolve the corresponding model transformations. This section further details our research methodology.

A. Study Participants

The experiment subjects consisted 15 master students in computer science, studying at the JKU Linz. They were selected out of 38 students. Those 15 students in addition of a past working experience, they have been working in parallel as part time programmer either in our computer science department in research projects or in companies as interns. This aimed to select a near experienced software developer.

B. Participants' Tasks

Our aim was to design a set of tasks that are close in scope and complexity to real tasks performed by developers, while being able to analyze the results effectively. In this experiment, we have designed the following tasks:

1) Creation of the metamodel and transforamtions (T0): As part of the *MDE* lecture, the students, as a first assignment, had to create a competition/tournament modeling language and write transformation scripts to transform a competition/tournament model into (I) a betting model (model-to-model transformation) and (II) a betting website (model-to-text transformation). We provided the betting metamodel as well as a description of a typical betting website.

The participants used EMF/Ecore to define their metamodels, and they had the choice between the transformation languages ETL or ATL (model-to-model) and EGL or Acceleo (model-to-text), all used in an Eclipse environment[1].

This first task acted as a preparation for the main part of our experiment which is comprised of the following tasks.

2) Metamodel evolution (T1): The first task for the participants was to evolve their metamodel, in particular by applying breaking changes that will have an impact on the transformations. However, we requested that each participant applies the following impacting changes (that we know are breaking changes from [14]):

(a) Delete elements (i.e., class/attributes/reference/parameter),
(b) Rename elements,
(c) Increase the multiplicity of an element from [0..1] to [*] (i.e., from single valued to multi valued),
(d) Move attributes/reference to another class,
(e) Push attributes/reference to subclasses.

The types of applied changes consisted of *atomic* and *complex* changes [12], [13]. *Atomic changes* are individual additions, removals, and updates of model elements. *Complex changes* consist of sequences of atomic changes combined together. In comparison to the atomic changes alone, complex changes include additional information on the interrelation of these atomic changes. For example, *push property* is a complex change where a property is moved from a superclass to subclasses. This is composed of multiple atomic changes, namely: one *delete* of property in the superclass and several *adds* of the property in the subclasses.

3) Impact analysis (T2): After evolving the metamodels, the participants had to identify manually the impacted parts of the transformations in order to co-evolve them afterwards.

4) Transformation co-evolution (T3): The final task was to co-evolve the impacted transformations, while documenting the applied resolutions. Furthermore, we have asked the participants to report on alternative resolutions that they can think of.

Table I: Data information.

Participants /Domains	Size of Metamodels in number of elements		Size of Transformations in number of LOC	
	Original	Evolved	Original	Evolved
P1:PokerTournament	89	85	377	355
P2:DanceTournament	86	84	153	153
P3: eSportsLeague	57	52	131	128
P4:SpaceRace	55	59	150	156
P5:SoccerEmModel	73	76	117	117
P6:languageolympics	84	83	194	203
P7:BasketballLeague	74	78	175	165
P8:contest	28	26	303	312
P9:Hackathon	57	55	101	101
P10:soccer_Tournament	63	54	308	315
P11:codingcontest	119	108	205	218
P12:nfl	109	113	576	573
P13:basketball	75	73	116	116
P14:golf_competition	49	49	148	148
P15:MPRPS	56	66	136	138

C. Data

Table I gives information about the used metamodels and transformations. It shows the size of the metamodels in number of elements and the size of the transformations in number of lines of code, for both original and evolved artifacts.

D. Research Questions

To investigate what is currently known about the co-evolution of transformations and to further learn about the manual co-evolution, we have defined the following research questions:

- RQ1: To what extend do transformation editors help in locating the impacted parts in a written transformation? This investigates whether providing an impact analysis is necessary or not when automating the co-evolution.
- RQ2: Do alternative resolutions often occur during co-evolution? This aims to investigate whether per impacted transformation, a unique resolution is often agreed upon, or alternative resolutions may be applied.
- RQ3: To what extent do the complex changes help during the co-evolution of transformations? This aims to investigate the benefit of detecting and having the knowledge of complex changes during the co-evolution.
- RQ4: What are the applied resolutions? As no catalog of resolutions exist, this aims to document a practically relevant set of resolutions that should be considered for

[1]http://www.eclipse.org/modeling/emf/, http://www.eclipse.org/epsilon/doc/etl/, https://eclipse.org/atl/, http://www.eclipse.org/epsilon/doc/egl/, https://www.eclipse.org/acceleo/

automation. It also allows us to assess the application frequency of each resolution.

- RQ5: To what extent are current automatic co-evolution techniques useful in our experiment? What are future work perspectives to enhance the state-of-the-art? Knowing the applied resolutions in our experiment, we can assess to what extent the existing techniques can handle the performed transformation co-evolution. This also aims to identify improvement aspects of current techniques that can be addressed in future work.

III. Results and Analysis

A. RQ1: To what extend do the transformation editors help in locating the impacted parts in a written transformation?

All our participants reported about the lack of support by the transformation languages/editors to locate impacted transformations, and also where exactly a transformation rule is impacted. Surprisingly, the used languages/editors did not even provide basic compilation support to highlight errors on the transformation rules themselves, as it is done in most of programming languages. To better illustrate the observed participants feedback, herein is an excerpt of what the participants reported on this lack of support for impact analysis.

"The ETL file showed no errors when opening it in the editor. The reason for this is that the ETL editor only checks for syntax errors and doesn't look into the referenced model at design time. Also the EGX program and the EGL templates showed no errors because of the same reasons."

As a consequence the participants performed a manual impact analysis to locate the impacted parts of the transformations. They used the knowledge of the applied metamodel changes. They also used the execution log of the transformations and the raised exceptions. This knowledge combination allowed the identification of the impacted parts of the transformations.

"When automatizing the transformation co-evolution, impact analysis should be included as part of the approach to release this burden from the user."

B. RQ2: Do alternative resolutions often occur during co-evolution?

To investigate the presence of alternative resolutions, we asked the participants to think about alternatives when possible. Only one participant highlighted an alternative repair for a transformation rule. As further investigation we checked how the different participants co-evolved transformations that were impacted by the same type of changes, e.g., how impacted transformations by delete changes are co-evolved by the different participants. We observed that the participants' transformations were co-evolved with different resolutions, which shows that alternative resolutions are indeed necessary to cope with various co-evolution scenarios.

For example, many participants applied a change multiplicity from 1..1 to 1..*, which in turn impacted multiple transformation rules. In some cases, a *for* iterator was introduced to access all values, and in some other cases, the operation *first()* was used to retrieve the first value of the collection. Another example where many participants applied different resolutions to co-evolve the same impact of move property changes. Depending on how the moved property was accessed (either from the source class or from the target class), the navigation path was either extended (e.g., from *obj.p* to *obj.ref.p*) or reduced (e.g., from *obj.ref.p* to *obj.p*).

Our observations thus confirm the presence of alternative resolutions, although not explicitly reported by most participants. However, as the set of alternative resolutions can be very large, identifying all alternative resolutions is a complex and challenging task which would require a larger scale study.

"As co-evolution is a creative task, a transformation can be co-evolved with alternative resolutions, which should be considered when automating the co-evolution."

C. RQ3: To what extent do the complex changes help during the co-evolution of transformations?

When analyzing the co-evolved transformations, we found that in most cases the applied resolutions are an intention to propagate the metamodel changes to the transformation rules. For example, every delete property *p* change led to a deletion of parts of the impacted transformations where *p* is used.

We observed that the applied complex changes, which had an impact on transformations, were indeed helpful during co-evolution, in comparison to atomic changes alone. The complex changes were useful in two ways. First, if only atomic changes are treated independently and not together as complex changes the applied resolutions would have been different. For example, considering a move property *p* as two independent changes delete property *p* and add property *p*, would lead to different resolutions as reported by the participants. Indeed, while the participants proposed to extend/reduce the navigation path accessing the property *p* since it is moved to another class, considering the independent changes delete property *p* and add property *p* would lead to delete the transformation part that uses *p*. Second, with atomic changes only, the user must spend extra effort in understanding the relations between atomic changes and identifying complex changes which have been already automated (e.g., in [13], [11]). In this experiment the same participants evolved the metamodel and co-evolved the transformations. Thus, the above issue did not arise. However, the above issue would arise if the evolution and co-evolution were performed by different developers, which is common in large or distributed software development projects.

"Complex changes helped in our experiment to co-evolve the transformations, and when automating the co-evolution, not only atomic changes but also complex changes should be considered."

D. RQ4: What are the applied resolutions?

Before assessing whether the current co-evolution techniques of transformations are suitable to automate the manual co-evolution w.r.t. the users' intent, i.e., whether they cover the

Table II: Applied resolutions during the co-evolution of transformations, with some examples from our participants.

	Applied Resolutions
▷[$R1$]	Remove an element used as part of an assignment in a transformation rule
▷[$R2$]	Remove part of a transformation (e.g., a line, a whole IF/loop/assignment instruction, etc.) rule
▷[$R3$]	Remove element used in a transformation template
▷[$R4$]	Rename elements
▷[$R5$]	Reduce navigation path of an accessed element
▷[$R6$]	Remove container call (e.g., *d.*`eContainer()`*.contestants* → *d.contestants*)
▷[$R7$]	Introduce a loop to iterate on a collection
▷[$R8$]	Extend navigation path of an accessed element
▷[$R9$]	Introduce operation first() on a collection (e.g., *lng.name* → *lng.name.first()*)
▷[$R10$]	Replace an element with another one (e.g., *"... + p.name"* → *"... + gr.title"*)
▷[$R11$]	Replace an element by a default value
▷[$R12$]	Replace an IF condition with another one
▷[$R13$]	Rename transformation rule name
▷[$R14$]	Introduce in If express testing different multiplicities

applied resolutions by our participants in this experiment. We must first list and categorize all manually applied resolutions.

Table II shows the applied resolutions by our participants and Figure 1 shows the frequency of their application in our experiment. Six resolutions were applied only once during the experiment while the other eight resolutions were applied at least six times and up to 104 times. Furthermore, Figure 2 shows the number of participants that applied each resolution. We observed that seven resolutions that were applied multiple times, were in fact applied by at least two different participants. This shows that those resolutions have an implicit consensus of their usefulness among our participants and they are likely to be agreed upon for automation.

"Table II represents an initial set of resolutions that should be considered for automation. In particular the resolutions that were applied multiple times and by multiple participants."

E. RQ5: To what extent are current automatic co-evolution techniques useful in our our experiment? What are future work perspectives to enhance the state-of-the-art?

To answer this research question, we have investigated the main co-evolution and refactoring approaches of transformations, namely: Alkhazi et al. [1], Kusel, et al. [14], Garcia et al. [4], Ruscio et al. [2], Garces et al. [3], and Mendez [15]. The threat to validity of missing a relevant approach and how we addressed this issue will be discussed in section V.

The first insight from investigating [1], [14], [4], [2], [3], [15] is that all those approaches include an impact analysis to identify the impacted transformations and do not delegate it to the users. Moreover, in order to assess to what extent the existing co-evolution approaches of transformations can

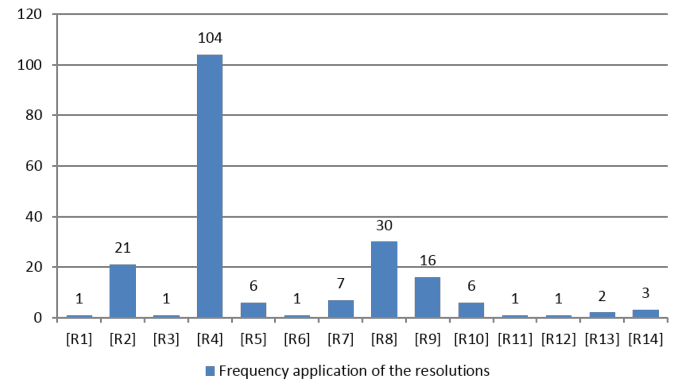

Figure 1: Number of applied resolutions by all participants in the experiment (Note the logarithmic scale).

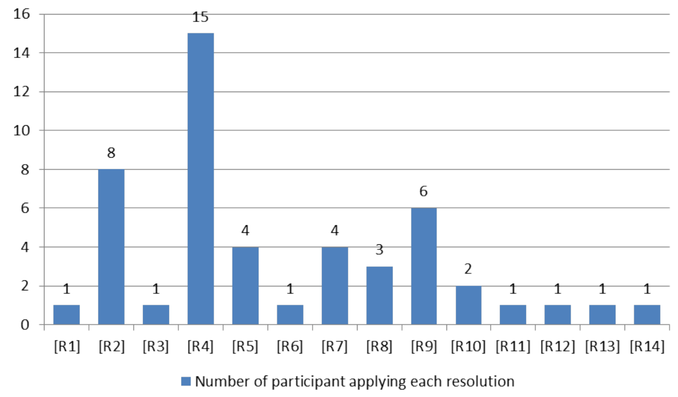

Figure 2: FrequencyRelevance.

handle automatically the manual co-evolution performed in our experiment, we investigated the supported resolutions in the existing approaches. Thus, we can see which resolutions from Table II are considered in the existing approaches [1], [14], [4], [2], [3], [15]. Table III illustrates the supported resolutions from Table II in the exiting approaches. We observed that no existing approach supports all manually applied resolutions in our experiment. This shows that the existing approaches cannot fully automate the performed co-evolution in out experiment, and thus, they are insufficient in the context of our experiment.

To alleviate this limitation, the missing resolutions can be first included in the existing approaches, which remains a matter of implementation. However, the main issue is that it is likely on other case studies that new resolutions would emerge. Thus, a future perspective would be to investigate empirically the applied resolutions on a larger set of data to converge on a common (and large) catalog of resolutions, similarly as it exist in refactoring. Nonetheless, since maintenance and co-evolution tasks are a creative engineering task, it is likely that there will always exist special cases that would require manual intervention from developers. We think that as a future perspective it would be interesting to distinguish atomic resolutions and complex resolutions analogously to

Table III: Existing techniques for transformation co-evolution and their mapping to resolutions in Table II.

Resol-utions	Alkhazi et al. [1]	Kusel, et al. [14]	Garcia et al. [4]	Ruscio et al. [2]	Garces et al. [3]	Mendez et al. [15]
[R1]	×	×	✓	×	×	×
[R2]	×	✓	✓	×	✓	✓
[R3]	×	✓	✓	✓	✓	×
[R4]	✓	✓	×	✓	✓	✓
[R5]	×	×	×	×	✓	×
[R6]	×	×	×	×	×	×
[R7]	×	×	✓	×	×	×
[R8]	×	✓	✓	✓	✓	✓
[R9]	×	×	✓	×	×	×
[R10]	×	×	×	✓	×	✓
[R11]	×	×	×	×	×	×
[R12]	×	×	×	×	×	×
[R13]	✓	×	×	×	×	×
[R14]	×	×	×	×	×	×

atomic and complex changes. It would be possible to propose a composition mechanism of resolutions (from atomic to complex resolutions) that can be guided by the user to co-evolve his/her transformations as near as possible to his/her intent. Therefore, different co-evolution scenarios could be handled which is not the case of the current existing techniques in our experiment.

"The currently investigated approaches showed to be insufficient in the context of our experiment due to a lack of resolution support."

IV. Discussion

Following the results analysis, we conducted a follow-up survey with the participants to collect their feedback on the experiment and more insight on the results. Out of the 15 participants, 10 answered in the survey. While all participants agreed upon the necessity to automate the co-evolution due to the difficulty of the manual co-evolution, only three participants preferred a fully automated co-evolution, while the rest rather preferred a semi-automated co-evolution. This was motivated by the fact that full automation would not guarantee that the transformations are co-evolved w.r.t. the users' intent, which requires users' intervention in the end.

We also asked the participants about which kind of a semi-automation would they prefer (multiple choices were allowed) between 1) *User decision* 2) *User confirmation* and 3) *User input*. *User decision* of what and how to apply resolutions came first with eight approvals, then *User confirmation* of the applied resolutions came as a second characteristic of a semi-automatic co-evolution with seven approvals. Providing concrete value as *User Input* came in third position with four approvals. Note that the two first choices User decision and User confirmation were preferred due to the fact that they require minimum intervention from the user while User input requires higher intervention effort providing values.

However, this is merely the beginning of an answer. To the best of our knowledge, it is little known which type of semi-automation would be more appropriate and in which context. In addition, a semi-automatic approach could combine the different types of user intervention, and an experiment study would be necessary to assess in which context a combination is more appropriate than another.

V. Threats to Validity

This section discusses the internal, external and conclusion threats to validity after Wohlin et. al. [27].

Internal Validity. In our experiment we selected master students as subject participants. Recent studies [22], [24] have shown that students can be valid subjects for experiments and that students are well representative when it comes to new developed tasks. Nonetheless, to further reduce this threat here we selected master students that already have 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. Finally, we did not inform the participants what we are investigating to avoid influencing them, however, we informed them that they will be graded for these tasks to fully ensure the participants' involvement.

External Validity. The used transformation languages in our experiment are ETL, ATL, EGL, and Acceleo. Thus, we cannot generalize our findings and observation to other transformation languages, especially w.r.t. the impact analysis observations where each transformation language differently handles differently the error detection. Nonetheless, most of the transformation languages share common concepts and similar grammar. In particular, the documented resolutions could also be applied to transformations written in other transformation languages. Moreover, the applied impacting metamodel changes did not cover all possible changes. However, they covered the main impacting changes that break the transformation rules [14]. Our goal was not to exhaustively apply all possible metamodel changes but to investigate how impacted transformations are co-evolved. Those threats are acceptable here.

Conclusion Validity. Only 15 students participated in this experiment. From a statistical point of view it would surely be better to include more participants both from academia and industry in order to gain a more precise insight and empirical evidence. However, results gained herein are sufficient to get an idea of what are the characteristics and difficulties to automate the co-evolution of transformations. Moreover, while assessing whether the existing co-evolution approaches of transformations are sufficient (in section III-E), we might have missed some approaches. To reduce this risk, for our 6 selected approaches, we used a snow ball strategy and Google-scholar to identify which new papers are citing the 6 selected papers. However, we did not perform a systematic literature review. Our goal herein was to assess whether the main known approaches within the community would be sufficient in our experiment to automate the transformations' manual co-evolution.

VI. Related Work

Maintenance is a major cost factor during the development of software and co-evolution support could potentially reduce that cost. To the best of our knowledge, so far no investigation explored manual co-evolution to identify and understand the aspects and characteristics for an efficient automatic co-evolution. There are some studies that focused on comparing transformation tools [10], [5] but not their maintenance and co-evolution. Perez et al. [18] who compared how transformations can be used for program refactoring activities. Rose et al. [20], [21] also compared how different transformation tools are used to co-evolve model instances. However, they investigated model co-evolution and not transformation co-evolution. Rose et al. [19] compared two approaches of transformation co-evolution and already then they highlighted the lack of support for impact analysis. Empirical work on transformation co-evolution was focused on very general questions [19] without using subjects to investigate the manual co-evolution of transformations. Our experiment consisted in running a manual co-evolution of transformations to better understand the requirements for an efficient automation for future approaches. It also investigated to what extent the existing techniques can automate the performed manual co-evolution.

VII. Conclusion

In this paper, we conducted and reported on an exploratory experiment with 15 participants investigating the evolution and the co-evolution of transformations. The results show that while transformation languages provide no support for impact analysis, the existing co-evolution approaches already support the user with an automatic impact analysis. However, we also observed that those existing approaches do not consider proposing a very large spectrum of alternative resolutions. In particular, among the 14 resolutions that occurred in our experiment, on average 4 (up to 6) out of the 14 were supported by the existing approaches. Finally, we highlighted the learned lessons and discussed potential future perspectives on how to improve the current state-of-the-art in transformations' co-evolution.

As a future work, we plan to reproduce the same experiment with other participants on the same original data to especially attempt to identify more alternative resolutions. We also plan to run two similar experiments on co-evolution of model instances and constraints due to metamodel evolution. Thus, it would then be very interesting to compare the co-evolution of the different artifacts altogether.

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

References

[1] B. Alkhazi, T. Ruas, M. Kessentini, M. Wimmer, and W. I. Grosky. Automated refactoring of atl model transformations: a search-based approach. In *The ACM/IEEE 19th MODELS*, pages 295–304, 2016.

[2] D. Di Ruscio, L. Iovino, and A. Pierantonio. A methodological approach for the coupled evolution of metamodels and atl transformations. In *ICMT*, pages 60–75. Springer, 2013.

[3] K. Garcés, J. M. Vara, F. Jouault, and E. Marcos. Adapting transformations to metamodel changes via external transformation composition. *Software & Systems Modeling*, 13(2):789–806, 2014.

[4] J. García, O. Diaz, and M. Azanza. Model transformation co-evolution: A semi-automatic approach. *SLE*, 7745:144–163, 2013.

[5] R. Grønmo, B. Møller-Pedersen, and G. K. Olsen. Comparison of three model transformation languages. In *ECMDA-FA*, pages 2–17. Springer, 2009.

[6] R. Hebig, D. E. Khelladi, and R. Bendraou. Surveying the corpus of model resolution strategies for metamodel evolution. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pages 135–142. IEEE, 2015.

[7] R. Hebig, D. E. Khelladi, and R. Bendraou. Approaches to co-evolution of metamodels and models: A survey. *IEEE Transactions on Software Engineering*, 43(5):396–414, 2017.

[8] Z. Hemel, L. C. Kats, D. M. Groenewegen, and E. Visser. Code generation by model transformation: a case study in transformation modularity. *Software and Systems Modeling*, 9(3):375–402, 2010.

[9] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of mde in industry. In *ICSE*, pages 471–480. ACM, 2011.

[10] E. Jakumeit, S. Buchwald, D. Wagelaar, L. Dan, Á. Hegedüs, M. Herrmannsdörfer, T. Horn, E. Kalnina, C. Krause, K. Lano, et al. A survey and comparison of transformation tools based on the transformation tool contest. *Science of computer programming*, 85:41–99, 2014.

[11] D. E. Khelladi, R. Bendraou, and M.-P. Gervais. Ad-room: a tool for automatic detection of refactorings in object-oriented models. In *ICSE Companion*, pages 617–620. ACM, 2016.

[12] D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, and M.-P. Gervais. Detecting complex changes during metamodel evolution. In *CAISE*, pages 263–278. Springer, 2015.

[13] D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, and M.-P. Gervais. Detecting complex changes and refactorings during (meta) model evolution. *Information Systems*, 2016.

[14] A. Kusel, J. Etzlstorfer, E. Kapsammer, W. Retschitzegger, W. Schwinger, and J. Schonbock. Consistent co-evolution of models and transformations. In *ACM/IEEE 18th MODELS*, pages 116–125, 2015.

[15] D. Mendez, A. Etien, A. Muller, and R. Casallas. Towards transformation migration after metamodel evolution. *ME Wokshop@MODELS*, 2010.

[16] T. Mens, G. Taentzer, and O. Runge. Analysing refactoring dependencies using graph transformation. *Software and Systems Modeling*, 6(3):269, 2007.

[17] T. Mens and P. Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

[18] J. Pérez, Y. Crespo, B. Hoffmann, and T. Mens. A case study to evaluate the suitability of graph transformation tools for program refactoring. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(3):183–199, 2010.

[19] L. Rose, A. Etien, D. Mendez, D. Kolovos, R. Paige, and F. Polack. Comparing model-metamodel and transformation-metamodel coevolution. In *International workshop on models and evolutions*, 2010.

[20] L. M. Rose, M. Herrmannsdoerfer, S. Mazanek, P. Van Gorp, S. Buchwald, T. Horn, E. Kalnina, A. Koch, K. Lano, B. Schätz, et al. Graph and model transformation tools for model migration. *Software & Systems Modeling*, 13(1):323–359, 2014.

[21] L. M. Rose, M. Herrmannsdoerfer, J. R. Williams, D. S. Kolovos, K. Garcés, R. F. Paige, and F. A. Polack. A comparison of model migration tools. In *MODELS*, pages 61–75. Springer, 2010.

[22] I. Salman, A. T. Misirli, and N. Juristo. Are students representatives of professionals in software engineering experiments? In *ICSE-Volume 1*, pages 666–676. IEEE Press, 2015.

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

[24] M. Svahnberg, A. Aurum, and C. Wohlin. Using students as subjects-an empirical evaluation. In *2nd ESEM*, pages 288–290. ACM, 2008.

[25] J.-P. Tolvanen and S. Kelly. Metaedit+: defining and using integrated domain-specific modeling languages. In *The 24th ACM SIGPLAN conference companion on OOPSLA*, pages 819–820, 2009.

[26] G. Wachsmuth. Metamodel adaptation and model co-adaptation. In *ECOOP*, pages 600–624. Springer, 2007.

[27] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.