Reality Check for Model Transformation Reuse: The ATL Transformation Zoo Case Study

A. Kusel[1], J. Schönböck[2], M. Wimmer[3],
W. Retschitzegger[1], W. Schwinger[1], and G. Kappel[3]

[1] Johannes Kepler University Linz, Austria
{firstname.lastname}@jku.at
[2] Upper Austrian University of Applied Sciences Hagenberg, Austria
{firstname.lastname}@fh-hagenberg.at
[3] Vienna University of Technology, Austria
{lastname}@big.tuwien.ac.at

Abstract. Model transformations play a major role in model-driven engineering. For increasing development productivity as well as quality of model transformations, different kinds of reuse mechanisms have been proposed. However, it remains unclear to which extent reuse mechanisms have made their way into practical application. Thus, this paper presents an empirical study on the ATL Transformation Zoo to analyze the application frequency of reuse mechanisms. For this, we developed a semi-automated process for extracting transformation projects from the ATL Transformation Zoo, which are classified and analyzed with respect to the application frequency of reuse mechanisms. Finally, limitations of current reuse mechanisms, which potentially hinder their practical applicability, are critically reflected, pointing out further research directions.

1 Introduction

Model-Driven Engineering (MDE) [15] proposes an active use of models to conduct the different phases of software development. Provided the fact that everything is a model, every systematic manipulation thereof may be considered a model transformation [16,18]. Consequently, model transformations are vital for MDE. Given their prominent role and their use in increasingly complex scenarios, appropriate reuse mechanisms are indispensable to increase development productivity as well as quality, e.g., in terms of maintainability of model transformations. To address this need, a plethora of reuse mechanisms has been proposed by the research community, cf., e.g., [1,2,3,4,7,8,10,11,13,17,21,22,23,25,26], to mention just a few. In [9], we have surveyed and categorized several different reuse mechanisms for model transformations by using a conceptual comparison framework. However, it remains still unclear, if at all and how often reuse mechanisms are employed in practical settings.

To shed some light on this area and to estimate the application frequency of current reuse mechanisms for model transformations, we performed a case study

based on a real-world transformation repository and its population. In particular, following the guidelines for conducting empirical explanatory case studies by Runeson and Höst [14], we analyzed the population of the Atlas Transformation Language (ATL) Transformation Zoo[4] (in the following denoted as "Zoo" for short) [6]. The Zoo has been chosen, because to the best of our knowledge, this repository is—at the time of writing—offering the most comprehensive collection of publicly available model transformations. Furthermore, the Zoo has been source for several previous studies concerning, e.g., the evaluation of model metrics [24], the validation of the results of metamodel matching tools [5], or the estimation of how end-users employ ATL in practice [19], to mention just a few. Thus, we consider the population of the Zoo as a representative set of model transformations. For analyzing the Zoo, we developed a semi-automated process for extracting transformation projects from the Zoo. The extracted transformation projects have then been classified and analyzed with respect to indicators for the application of reuse mechanisms. The results show that up to now reuse mechanisms are rarely used in practice. Thus, we conclude by discussing potential barriers that might harm the practical applicability, thereby pointing to further research topics.

Outline. Section 2 discusses the basic setup of the case study and gives an impression on the transformations that are available in the Zoo. The case study as well as the results thereof are presented in Section 3, whereby Section 4 critically reflects the results and discusses threats to validity, before Section 5 concludes the paper.

2 Case Study Setup

To estimate the application frequency of current reuse mechanisms for model transformations, we conducted an empirical explanatory case study in order to analyze the population of the Zoo. The study was performed to quantitatively assess the application frequency of reuse mechanisms in model transformations of the Zoo. More specifically, we aimed at answering the following research question:

> *With which frequency are reuse mechanisms currently applied in model transformation projects?*

2.1 Case Study Design

For performing the analysis of the Zoo's population, we extracted all transformation projects from the Zoo's website[5]. To reason about the Zoo's population, not only the transformation definitions are required, but also accompanying artifacts such as the input and output metamodels, launch configurations, and build scripts. For instance, the metamodels are of interest to relate their size

[4] `http://www.eclipse.org/m2m/atl/atlTransformations`

[5] The complete data of this snapshot is available on our project website `http://www.modeltransformation.net`

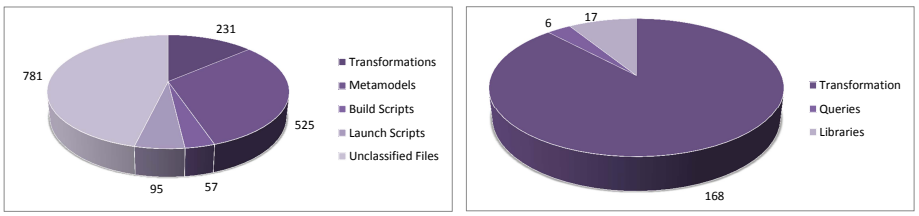
and structure to the size and structure of model transformations. The launch configuration files and build scripts for running the transformations are an important source to reason about the execution processes of the transformations, e.g., how a set of transformations interact.

The Zoo provides a collection of 103 different transformation projects, mostly provided as .zip archives, containing 1689 files in total. Before starting the automated analysis of the transformation projects in a subsequent step, the relevant files (transformations, metamodels, launch configurations, and build scripts) have been extracted from the .zip archives on basis of their file extensions. Consequently, we selected 873 out of the 1689 files: 231 ATL transformations (`.atl` files), 525 metamodels (`.ecore` and `.km3` files), 57 build scripts (`.build` or `.xml` files), and 95 launch scripts (`.launch` files). The remaining 781 files, being, e.g., readme files for documentation or test input/output models, have not been considered in this case study (cf. Fig. 1(a)). Furthermore, since transformations have been reused in different projects, duplicates may exist. In particular, we regarded two transformations as duplicate, if they have the same name and identical metric values for their intrinsic properties, i.e., the same number of rules and helpers. In this respect, 40 duplicates have been removed resulting in 191 transformations for further investigation. The remaining transformations may be further divided into three different kinds of ATL transformations. First, a transformation may either be (*i*) a model-to-model transformation (168), (*ii*) a library of reusable helpers (17), which are importable to other transformations, or (*iii*) queries (6), which derive information from models by using Object Constraint Language[6] (OCL) expressions, e.g., to select a set of model elements from an input model, as depicted in Fig. 1(b).

2.2 Characteristics of the Zoo

To give an impression on the complexity of the transformation tasks supported, we list in the following some meta-information about the transformations. First, the transformation size ranges from a minimum of 1 rule to a maximum of 84 rules. Concerning the numbers of input and output metamodels, the majority are 1-to-1 transformations, but there is also a small amount of 1-to-n (2 transformations), n-to-1 (12 transformations), and n-to-m transformations (2 transforma-

[6] http://www.omg.org/spec/OCL



(a) Distribution of file types (b) Distribution of transformation types

Fig. 1. Overview on the ATL Transformation Zoo.

tions) in the Zoo available. The metamodels used in the transformations range from small metamodels (below 10 meta-classes) to large metamodels (about 280 meta-classes). The languages represented by the metamodels range from modeling languages such as UML, QVT, and OCL over markup languages such as HTML and SVG to general-purpose programming languages such as Java or domain-specific languages such as BibTeX. Finally, since support for different reuse mechanisms has been successively added over the last years, e.g., functions have been introduced with the first version of ATL (2005), whereas inheritance (2006) and superimposition (2007) have been introduced in subsequent versions, the submission date to the Zoo is of interest.

3 Analysis of the ATL Transformation Zoo

After introducing the setup of the case study, it is described to which extent the reuse mechanisms available for ATL are practically applied by transformations in the Zoo, whereby we distinguish between a fully automatic detection and a semi-automatic detection. For each reuse mechanism, we (*i*) shortly characterize the according reuse mechanism, (*ii*) discuss, how it might be (semi-)automatically detected, (*iii*) analyze the results, and finally (*iv*) provide a critical discussion thereof. The results are summarized in Table 1. For calculating the relative application frequency, we refer to the ratio between applications detected and total amount of model transformations (168). However, for the application frequency of transformation chains, we employ the ratio between applications and total amount of transformation projects (103), because a chain is not tailored to one transformation, but to a complete transformation project.

Table 1. Reuse mechanisms' applications and frequency.

	Reuse Mechanism	Total Number of Applications	Relative Application Frequency
Automatic Detection	*Functions*	134	79%
	Rule Inheritance	6	4%
	Superimposition	0	0%
	HOTs	7	4%
	Transformation Orchestration	11	11%
Semi-Automatic Detection	*TPLs*	0	0%
	External DSL	0	0%
	Generic Transformations	0	0%

3.1 Automatically Detected Reuse Mechanisms

Reuse mechanisms, whose applications might be detected automatically include functions, inheritance, superimposition, higher-order transformations (HOTs), and transformation orchestration, as described in the following.

Functions. As well-known from procedural programming languages, functions in transformation languages provide means to extract and to reuse recurring transformation logic. In ATL, functions are called `helpers` and are defined in OCL. The application frequency of functions in ATL transformations may be detected automatically by querying, if `helpers` are contained in a transformation as indicated by the following OCL query.

```
Transformation.allInstances() -> select(t|t.helpers.notEmpty())
```

When analyzing the result, it may be seen that helpers are used in nearly 80% of the inspected transformations (cf. Table 1) and the higher the amount of rules within a transformation, the higher the amount of helpers is (cf. Fig. 2(a)). This might be due to the fact that helpers are included, since the very first version of ATL and also because functions are a well-known reuse mechanism from traditional software engineering. Furthermore, functions might be that popular, because they are expressed in OCL, and consequently, there is no further learning curve for the transformation designer, since she is typically familiar with OCL.

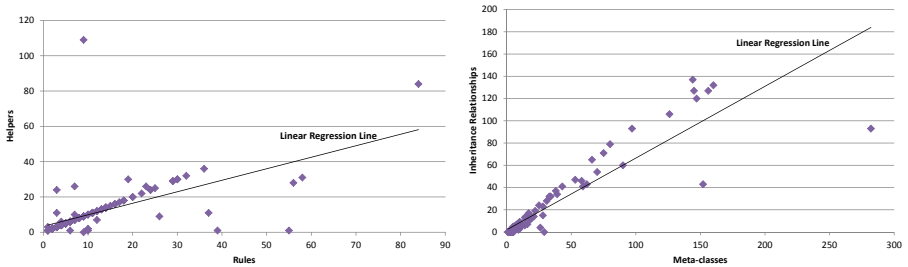
Rule Inheritance. Inheritance between meta-classes in metamodels necessitates the usage of inheritance between transformation rules to avoid code duplication, e.g., duplicate assignments. The application frequency of rule inheritance may be automatically detected by searching for rules that extend other rules, which is indicated by the reference `Rule.superRule` in the ATL metamodel and exploited in the following OCL query.

```
Transformation.allInstances() -> select(t|t.rules -> exists(r|r.
    ↪superRule <> OclUndefined))
```

Although, a tremendous amount of metamodels of the Zoo employs inheritance (around 75%), rule inheritance is rarely used in the Zoo (6 applications, only),[7] whereby there is a strong correlation (correlation coefficient about 0.90) between the amount of meta-classes and the amount of inheritance relationships (cf. Fig. 2(b)). Consequently, rule inheritance would be especially beneficial for large metamodels. However, most surprisingly, inheritance between rules has been used rather by middle-sized transformations. A reason for the poor adoption of inheritance in model transformations might be that the support for inheritance in ATL is still limited, e.g., the declarative part of ATL is considered in rule inheritance [27], only. Furthermore, there is only limited support for static semantic checks, aggravating the correct application of rule inheritance.

Module Import. Module import allows to build the union of transformation rules from different model transformations. Thereby, rules or helpers may be

[7] It has to be noted that inheritance has been introduced in the ATL 2006 compiler. This induces that rule inheritance may have been theoretically employed in about 60% of all transformations.



(a) Helpers vs. rules (b) Classes vs. inheritance relationships

Fig. 2. Overview on relationships between different element types.

redefined, i.e., a rule or a function may be replaced by a new one, and additional rules and functions may be added. This concept is known in ATL as superimposition [25]. To automatically recognize superimposition, the launch scripts have been analyzed, since superimposition is introduced at load-time in ATL (cf. OCL query below). Thereby, superimposition is assumed to be used, if a launch script contains an entry with a key `Superimpose` and a non-empty value.

```
LaunchScript.allInstances() -> select(ls|ls.entries -> exists(entry|
    ↪entry.key = 'Superimpose' and entry.value <> OclUndefined))
```

Although superimposition has been introduced in 2007 already, currently no transformation in the Zoo applies this reuse mechanism. A reason for this might be that this mechanism is rather coarse-grained, i.e., rules that should be redefined must be redefined from scratch without the possibility of reusing parts of the refined rule. Consequently, it would be beneficial, if superimposition could be combined with inheritance. Unfortunately, superimposition is not compatible with inheritance, i.e., the rule inheritance hierarchy is broken, if a superrule is redefined with superimposition. Furthermore, ATL imports modules at load-time, whereas numerous other transformation languages import modules at compile-time [9], entailing the advantage that static checks may be applied.

Higher Order Transformation (HOT). HOTs are model transformations that either take a model transformation as input, produce a model transformation as output, or do both and may thus, be used for transformation synthesis, transformation analysis, transformation (de-)composition, or transformation modification [21]. For automatically detecting the usage of a HOT, one has to analyze, whether the input metamodel and/or the output metamodel of a transformation is of type ATL as done by the following OCL query.

```
Transformation.allInstances() -> select(t| t.models -> exists(m|m.name =
    ↪ 'ATL'))
```

When analyzing the transformations of the Zoo, one may find that the application frequency for HOTs is around 4%. HOTs are available in the Zoo, especially for (*i*) transformation synthesis, e.g., to produce from metamodels a copying transformation for their models and for (*ii*) transformation modification, e.g., to enrich ATL transformations by adding debugging functionality or tracing capabilities. The low application frequency of 4% may result from the challenging development of HOTs [20] and from the specialized application cases.

Transformation Orchestration. Transformation orchestration is used to reuse transformations in the large, i.e., whole transformations at once. For orchestrating model transformations, build files on basis of ANT[8] may be used. Therefore, the automatic detection of transformation orchestrations relies on the recognition, if more than one task for executing a transformation is defined in the build script as formalized by the following OCL query.

```
BuildScript.allInstances() -> select(bs| bs.tasks -> select(t|t.name = '
    ↪atl.launch' or t.name = 'am3.atl') -> size() > 1)
```

[8] `http://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Tools#ATL_ant_tasks`

When investigating the Zoo, transformation orchestration is used in around 11% of the projects, especially in transformation projects that establish bridges between technical spaces. Some transformation chains are quite complex such as employing not only sequences, but also loops of transformation executions, i.e., a transformation is employed for an arbitrary sized collection of models. In the largest model transformation chain, nine transformations are involved. Finally, also HOTs are used in the chains to produce transformations on-the-fly that are applied directly in the later phases of the transformation process.

3.2 Semi-Automatically Detected Reuse Mechanisms

Besides those reuse mechanisms that might be detected fully automatically, some reuse mechanisms allow for a semi-automatic detection, only, including transformation product lines (TPLs), external domain specific languages (DSLs), and generic transformations (cf. [9] for details). In this context, TPLs allow to configure a transformation externally, i.e., to use, e.g., a feature model to configure a transformation and by this to reuse the already predefined transformation rules. An example for this might be a Class2Relational transformation, which allows to follow different object-relational mapping strategies, which might be configurable by a feature model. External DSLs on top of ATL allow to simplify the specification of recurring transformation logic by dedicated language constructs, which finally get translated into ATL code. Finally, generic transformations allow to parameterize transformation logic with types, and thus, allow to decouple transformation logic from concrete metamodel types.

Current best practice in ATL to implement a TPL is to use an additional input model to configure a transformation. Thus, we selected each transformation having more than one input model as a potential candidate that has to be inspected manually.

```
Transformation.allInstances() -> select(t|t.inModels -> size() > 1)
```

External DSLs and generic transformations typically employ a HOT that either generates a new ATL transformation (external DSL) or that rewrites an existing one (generic transformations). Consequently, hints for the application of these reuse mechanisms may be detected by analyzing, if the ATL metamodel is used as the target metamodel of a transformation (cf. OCL query below). The resulting hints need to be verified by manual inspection of the transformations.

```
Transformation.allInstances() -> select(t| t.outModels -> exists(m|m.
    ↪name = 'ATL'))
```

Although candidate transformations for these reuse mechanisms have been detected, the manual inspection thereof showed that none of these reuse mechanisms have been applied in the transformations of the Zoo, which might be due to the fact that those reuse mechanisms just emerged recently and are thus, not reflected in the Zoo. Finally, please note that we did not investigate internal DSLs defined for ATL (e.g., HNL [2], ATL4pros [12]), and reflection as provided by Mistral [8], because these approaches require for a modified ATL execution environment and the transformations contained in the Zoo are executable with the official distribution of ATL, only.

4 Discussion

We now present (*i*) a critical discussion of the results and (*ii*) we elaborate on several factors that may jeopardize the validity of our results.

Well-known Reuse Mechanisms Made their Way into Practice. With respect to the posed research question, one may see that the frequency of the application of reuse mechanisms varies strongly between the different reuse mechanisms. Helpers are frequently used in transformations. This seems quite natural, because (*i*) factorization of recurring logic to functions is well-known from procedural programming languages, (*ii*) OCL, which is mainly used to define helpers, is a well-known language for transformation developers, and (*iii*) helpers have been provided from the early stages of ATL – thus, it is also well-documented in the ATL user guide and well-demonstrated by several examples. Furthermore, at least some transformations apply rule inheritance, which is comparable to inheritance in object-oriented programming languages. Finally, orchestration is also a common and well-understood reuse mechanism in software engineering and has achieved practical application.

Reuse Occurs in a Narrow Scope. By further investigating the applied reuse mechanisms, it may be seen that reuse occurs most often within a single transformation, only, i.e., reuse across transformation boundaries is performed rarely. Thus, it may be concluded that reuse mechanisms that have a direct and instant benefit for the transformation developer, when creating a single transformation are applied more frequently. Other reuse mechanisms such as TPLs, generic transformations, and external DSLs, which unfold their full potential over the time and require more complex abstraction and specialization mechanisms, still have to wait for their frequent application.

Challenging Abstraction/Specialization may Hamper Application. Any reusable artifact needs abstraction as well as specialization to be adapted to the current context. However, the abstraction of reusable artifacts is often challenging. This applies especially to HOTs as also stated by Tisi et al. [20], where the user must be familiar with the abstract syntax of the transformation language. In case of generic transformations, specialization requires that mappings between the metamodels of the transformation to reuse and the new transformation have to be defined by the transformation designer in order to overcome heterogeneities between the involved metamodels.

Threats to Validity. *Internal Validity: Are There Factors, Which Might Affect the Results in the Context of ATL?* Applications of superimposition may have not been found, because of missing launch scripts. Sometimes screenshots are provided, only that may not be processed automatically to detect applications of reuse mechanisms. The same holds for missing build scripts in case of transformation chains or chains that are executed manually or by Java programs.

The results may be biased, because only ATL transformations residing in the Zoo have been analyzed. Latest trends in transformation reuse may have not been reflected, since the latest transformations stem from October 2010.

External Validity: To What Extent is it Possible to Generalize the Findings? So far, we cannot claim any results outside the context of the Zoo. Nevertheless, the analysis methods may be applied to arbitrary transformation repositories to compute the frequency of the employed reuse mechanisms. Thus, replaying the presented case study for other transformation languages and repositories should enable the possibility of reasoning about the reuse mechanism applications for those languages/repositories as well.

5 Conclusion

In this paper, we reported on a case study for analyzing the Zoo's population with respect to the application frequency of reuse mechanisms. For this, we developed a framework for analyzing the population in a semi-automated way. This framework is publicly available and is customizable to investigate other language usage aspects in the future as well.

We see the following topics as possible next steps of this work. First, by having transformations in the Zoo identified that are not using rule inheritance, although the source and target metamodels are heavily using inheritance between meta-classes, would allow to experiment with automated refactorings [28] for improving the transformations' designs. Second, we plan to explore additional transformations that are publicly available but outside of the Zoo. Finally, we also want to expand our work to other transformation languages that offer reuse mechanisms such as QVT.

Acknowledgements. This work has been funded by BMVIT under grants FFG BRIDGE 832160 and FFG FIT-IT 825070 and 829598, FFG Basisprogramm 838181, and by ÖAD under grant AR18/2013 and UA07/2013. We would like to thank Marcel F. van Amstel for providing us the ATL2Metrics transformations which has been the basis for computing several metrics for the Zoo's population.

References

1. J. Cuadrado, E. Guerra, and J. de Lara. Generic Model Transformations: Write Once, Reuse Everywhere. In *ICMT'11*, pages 62–77. Springer, 2011.
2. J. Cuadrado, F. Jouault, J. García Molina, and J. Bézivin. Experiments with a High-Level Navigation Language. In *ICMT'09*, pages 229–238. Springer, 2009.
3. J. Cuadrado and J. G. Molina. A Model-Based Approach to Families of Embedded Domain-Specific Languages. *IEEE TSE*, 35(6):825–840, 2009.
4. M. Del Fabro and P. Valduriez. Towards the Efficient Development of Model Transformations using Model Weaving and Matching Transformations. *SoSyM*, 8(3):305–324, 2009.
5. K. Garcés, W. Kling, and F. Jouault. Automatizing the Evaluation of Model Matching Systems. In *Workshop on Matching and Meaning*, 2010.
6. F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proc. of the Int. Conf. on Satellite Events at the MoDELS*, MoDELS'05, pages 128–138, 2006.

7. A. Kleppe. MCC: A Model Transformation Environment. In *ECMDA-FA'06*, pages 173–187. Springer, 2006.
8. I. Kurtev. Application of Reflection in a Model Transformation Language. *SoSyM*, 9(3):311–333, 2010.
9. A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger. Reuse in model-to-model transformation languages: Are we there yet? *SoSyM*, pages 1–31, 2013. online first.
10. E. Legros, C. Amelunxen, F. Klar, and A. Schürr. Generic and Reflective Graph Transformations for Checking and Enforcement of Modeling Guidelines. *Visual Language Computing*, 20(4):252–268, 2009.
11. J. Oldevik. Transformation Composition Modelling Framework. In *DAIS'05*, pages 108–114. Springer, 2005.
12. A. Randak, S. Martínez, and M. Wimmer. Extending ATL for Native UML Profile Support: An Experience Report. In *MtATL'11*, pages 49–62, 2011.
13. J. E. Rivera, D. Ruiz-Gonzalez, F. Lopez-Romero, J. Bautista, and A. Vallecillo. Orchestrating ATL Model Transformations. In *MtATL'09*, pages 34–46, 2009.
14. P. Runeson and M. Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Soft. Eng.*, 14(2):131–164, 2009.
15. D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.
16. S. Sendall and W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45, 2003.
17. M. Sijtema. Introducing Variability Rules in ATL for Managing Variability in MDE-based Product Lines. In *MtATL'10*, pages 39–49, 2010.
18. E. Syriani. *A Multi-Paradigm Foundation for Model Transformation Language Engineering*. Ph.D. Thesis, McGill University, February 2011.
19. R. Tairas and J. Cabot. Corpus-based analysis of domain-specific languages. *SoSyM*, pages 1–16, 2013. online first.
20. M. Tisi, J. Cabot, and F. Jouault. Improving Higher-Order Transformations Support in ATL. In *ICMT'10*, pages 215–229. Springer, 2010.
21. M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the Use of Higher-Order Model Transformations. In *ECMDA-FA'09*, pages 18–33. Springer, 2009.
22. B. Vanhooff, D. Ayed, S. Van Baelen, W. Joosen, and Y. Berbers. UniTI: A Unified Transformation Infrastructure. In *MODELS'07*, pages 31–45. Springer, 2007.
23. D. Varró and A. Pataricza. Generic and Meta-Transformations for Model Transformation Engineering. In *UML'04*, pages 290–304. Springer, 2004.
24. E. Vépa, J. Bézivin, H. Bruneliére, and F. Jouault. Measuring Model Repositories. In *Workshop on Model Size Metrics (MSM'06)*, 2006.
25. D. Wagelaar, R. Van Der Straeten, and D. Deridder. Module Superimposition: A Composition Technique for Rule-based Model Transformation Languages. *SoSyM*, 9(3):285–309, 2010.
26. M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Surviving the Heterogeneity Jungle with Composite Mapping Operators. In *ICMT'10*, pages 260–275. Springer, 2010.
27. M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, W. Schwinger, D. Kolovos, R. Paige, M. Lauder, A. Schürr, and D. Wagelaar. Surveying Rule Inheritance in Model-to-Model Transformation Languages. *JOT*, 11(2):3:1–46, 2012.
28. M. Wimmer, S. M. Perez, F. Jouault, and J. Cabot. A Catalogue of Refactorings for Model-to-Model Transformations. *JOT*, 11(2):2:1–40, 2012.