Software and Systems Modeling (2021) 20:469–503
https://doi.org/10.1007/s10270-020-00815-4

REGULAR PAPER

Claimed advantages and disadvantages of (dedicated) model transformation languages: a systematic literature review

Stefan Götz[1] · Matthias Tichy[2] · Raffaela Groner[2]

Received: 20 November 2019 / Revised: 9 June 2020 / Accepted: 16 June 2020 / Published online: 14 July 2020

Abstract
There exists a plethora of claims about the advantages and disadvantages of model transformation languages compared to general-purpose programming languages. With this work, we aim to create an overview over these claims in the literature and systematize evidence thereof. For this purpose, we conducted a systematic literature review by following a systematic process for searching and selecting relevant publications and extracting data. We selected a total of 58 publications, categorized claims about model transformation languages into 14 separate groups and conceived a representation to track claims and evidence through the literature. From our results, we conclude that: (i) the current literature claims many advantages of model transformation languages but also points towards certain deficits and (ii) there is insufficient evidence for claimed advantages and disadvantages and (iii) there is a lack of research interest into the verification of claims.

Keywords Model transformation language · DSL · Model transformation · MDSE · Advantages · Disadvantages

1 Introduction

Ever since the dawn of model-driven engineering at the beginning of the century, model transformations, supported by dedicated transformation languages [31], have been an integral part of model-driven development. Model transformation languages (MTLs), being domain-specific languages, have ever since been associated with advantages in areas like productivity, expressiveness and comprehensibility compared to general-purpose programming languages (GPLs) [50,55,60]. Such claims are reiterated time and time again in the literature, often without any actual evidence. Nowadays, such an abundance of claims runs through the whole literature body that one can be forgiven when losing track of which claims verifiably apply and which are still purely visionary.

The **goal** of this study is to identify and categorize claims about advantages and disadvantages of model transformation languages made throughout the literature and to gather available evidence thereof. We do not intend to provide a complete overview over the current state of the art in research. For this purpose, we performed a systematic review of claims and evidence in the literature.

The main **contributions** of our study are:

- a systematic review and overview over the advantages and disadvantages of model transformation languages as claimed in the literature;
- insights into the state of verification of aforementioned advantages and disadvantages;

This study is intended for researchers to (i) raise awareness for the current state of research and (ii) incentivise further research in areas where we identified gaps. The study can also be of interest to practitioners who wish to gain an overview over what research claims about MTLs compared to a practitioners view of the matter.

To systematize information from the literature, we performed a systematic literature review [14,41] based on the research questions we defined (see Sect. 3.1). As a first step,

Communicated by Alfonso Pierantonio.

✉ Stefan Götz
stefan.goetz@uni-ulm.de

Matthias Tichy
matthias.tichy@uni-ulm.de

Raffaela Groner
raffaela.groner@uni-ulm.de

1 Ulm University, 89081 Ulm, Germany

2 Ulm University, 80901 Ulm, Germany

during the review we selected 58 publications from which to extract claims and evidence for advantages and disadvantages of model transformation languages. Afterwards, we categorized claims and systematized the evidence to produce (i) a categorization of claimed advantages and disadvantages into 15 separate categories (namely *analysability*, *comprehensibility*, *conciseness*, *debugging*, *ease of writing a transformation*, *expressiveness*, *extendability*, *just better*, *learnability*, *performance*, *productivity*, *reuse and maintainability*, *tool support*, *semantics and verification*, *versatility*) and (ii) a systematic representation of which claims are verified through what means. From our results, we conclude that:

1. The current literature claims many advantages and disadvantages of model transformation languages.
2. A large portion of claims are very broad.
3. There is insufficient or no evidence for a large portion of claims.
4. There is a number of claims that originate in claims about DSLs without proper evidence why they hold for MTLs too.
5. There is a lack of research interest in evaluation and especially verification of claimed advantages and disadvantages.

We hope our results can provide an overview over what MTLs are envisioned to achieve, what current research suggests they do and where further research to validate the claimed properties is necessary.

The remainder of this paper is structured as follows: Sect. 2 introduces the background of this research, model-driven engineering and model transformation languages. In Sect. 3, we will detail the methodology used for the conducted literature review. We present our findings in Sect. 4. Afterwards, in Sect. 5, we discuss the results of our findings. This section will also include propositions for much needed validation of claims about model transformation languages synthesized from the literature review. Section 6 contains information about related work, and in Sect. 7 potential threats to the validity of this research are discussed. Lastly, Sect. 8 draws a conclusion for our research.

2 Background

In this section, we provide the necessary background for our study and explain the context in which our study integrates.

2.1 Model-driven engineering

In 2001, the Object Management Group published the software design approach called *Model-Driven Architecture* [52] as a means to cope with the ever-growing complexity of software systems. MDA placed models at the centre of development rather than using them as mere documentation artefacts. The approach envisions an automated, continuous specialization from abstract models towards code. Starting with the so-called *Computation Independent Models* (CIMs), each specialization step should provide the models with more specific information about the intended system, transforming them from *CIM* into *Platform Independent Models* (PIMs) and then into *Platform Specific Models* (PSMs) and finally into production ready source code.

The different abstraction levels were designed to enable practitioners to be as platform, system and language independent as possible. The notion of using models as the central artefact during development is what is commonly referred to as *Model-Driven (Software-) Engineering* (MDE/MDSE) or *Model-Based (Software-) Engineering* (MBE/MBSE) [20].

The structure of a model is defined by a so-called meta-model whose structure is then also defined by meta-models of their own.

2.2 Domain-specific languages

"A domain-specific language (DSL) provides a notation tailored towards an application domain and is based on relevant concepts and features of that domain" [61]. The idea behind this design philosophy is to increase expressiveness and ease of use through more specific syntax. As such, DSLs provide an auspicious alternative for solving tasks associated with a specific domain. Representative DSLs include *HTML* for designing Web pages or *SQL* for database querying and manipulation.

2.3 Model transformation languages

Models are transformed into different models of the same or a different meta-model via the so-called *model transformations*. Driven by the appeal of DSLs, a plethora of dedicated MTLs have been introduced since the emergence of MDE as a software development approach [3,7,38,43]. Unlike general-purpose programming languages, MTLs are designed for the sole purpose of enabling developers to transform models. As a result, model transformation languages provide explicit language constructs for tasks performed during model transformation such as model matching. Similar to GPLs, model transformation languages can differ vastly in several aspects, starting with features that can be found in GPLs as well like language paradigm and typing all the way to transformation-specific features such as directionality [22]. There are numerous of features that can be used to distinguish model transformation languages from one another. For a complete classification of these features, please refer to Kahani et al. [39], Mens and Gorp [49] or Czarnecki and Helsen [22].

Model transformation languages, being DSLs, promise dedicated syntax tailored to enhance the development of model transformations.

3 Methodology

Our review procedures are based on the descriptions of literature and mapping reviews from Boot, Sutton and Papaioannou [14]. First of all, a protocol for the review was defined. The protocol, as defined in Boot, Sutton and Papaioannou [14], describes (I) the research background (see Sect. 2), (II) the objective of the review and review questions (see Sect. 3.1), (III) the search strategy (see Sect. 3.2), (IV) selection criteria for the studies (see Sect. 3.3), (V) a quality assessment checklist and procedures (see Sect. 3.4), (VI) the strategy for data extraction and (VII) a description of the planned synthesis procedures (see Sect. 3.5). A complete overview of all steps of our literature review can be found in Sect. 1.

The remainder of this section will describe in detail each of the introduced protocol elements, with the exemption of the research background which we already covered in Sect. 2.

3.1 Objective and research questions

To formulate the objective as well as to derive the research questions for our review, we first applied the *Goal-Question-Metric* approach [11] which splits the overall goal into four separate concerns, namely *purpose*, *issue*, *object* and *viewpoint*.

Purpose	Find and categorize
Issue	claims of and evidence for advantages and disadvantages
Object	of model transformation languages
Viewpoint	from the standpoint of researchers and practitioners.

Based on the described goal, we then extracted the two main research questions for our literature review:

RQ1	What advantages and disadvantages of model transformation languages are claimed in the literature?
RQ2	What advantages and disadvantages of model transformation languages are validated through empirical studies or by other means?

The aim of *RQ1* is to provide an extensive overview over what kinds of advantages or disadvantages are explicitly attributed to using dedicated model transformation languages compared to using general-purpose programming languages. We consider such an overview to be necessary, because the number of claims and their repetition in the literature to date makes it difficult to keep track of which claims verifiably apply and which are still purely visionary. Naturally to be able to distinguish between substantiated and unsubstantiated claims, it is also required to record which claims are supported by evidence. With *RQ2*, we aim to do exactly that. Combining the results of *RQ1* and *RQ2* then makes it possible to determine if, and how, a positive or negative claim about MTLs is verified. Additionally, this also enables us to identify those claims that have yet to be investigated.

3.2 Search strategy

Our search strategy consists of seven consecutive steps. A visual overview of the complete search process is shown in Fig. 3. The figure visualizes steps *Database search* to *Snowballing* from Fig. 1 in more detail.

In the first step, we defined the search string to be used for automatic database searches. For this, we identified major terms concerning our research questions. Each new term was made more specific than the previous one. The resulting terms and justifications for including them were:

- *Model-driven engineering* The overall context we are concerned with. This was included to ensure only papers from the relevant context were found.
- *Model transformation* The more specific context we are concerned with.
- *Model transformation language* Since our focus is on the languages to express model transformations.

We used a thesaurus to identify relevant synonyms for each term in order to enhance our search string. In addition, we included one representative model transformation language with graphical syntax, one imperative language, one declarative language and one hybrid language as well as the term *domain-specific language* and its synonyms. The selection of the representative languages was made on the basis of their widespread use, active development and in the case of *QVT* because it is the standard for model transformations adopted by the Object Management Group. All these additional terms were included as synonyms for the *model transformation language* term.

We dropped the terms *advantage* and *disadvantage* after initial searches, because they resulted in a too narrow of a result set which excluded key publications [29,33] manually identified by the authors.

To combine all keywords, we followed the advice of Kofod-Petersen [42] to use the Boolean ($\vee$) to group together synonyms and the Boolean ($\wedge$) to link our major term groups.

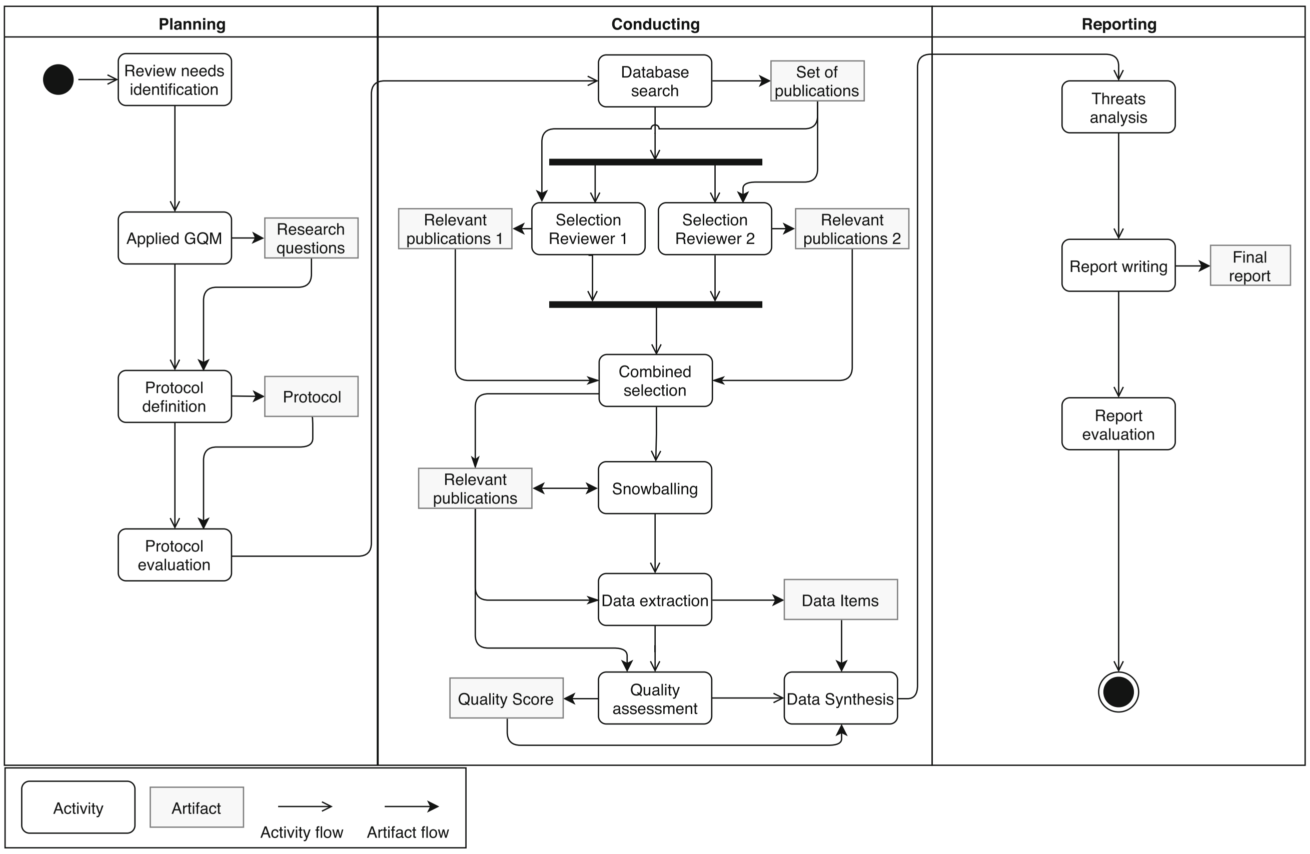

Fig. 1 Protocol overview

(Model Driven Engineering $\vee$ MDE $\vee$ Model Based Engineering $\vee$
MBE $\vee$ Model Driven Development $\vee$ MDD $\vee$
Model Driven Software Engineering $\vee$ MDSE $\vee$
Model Driven Software Development $\vee$
MDSE $\vee$ Model-Driven Software Development $\vee$
Model-Driven Engineering $\vee$ Model-Based Engineering $\vee$
Model-Driven Software Engineering)
$\wedge$
(Model Transformation $\vee$ Transformation $\vee$
Model Transformations $\vee$ Transformations)
$\wedge$
(Model Transformation Language $\vee$ Transformation Language $\vee$
ATL $\vee$ Henshin $\vee$ QVT $\vee$ TL $\vee$
Transformation Languages $\vee$ DSL $\vee$ domain specific language $\vee$
Model Transformation Languages)

Fig. 2 Search string used for automatic database searches

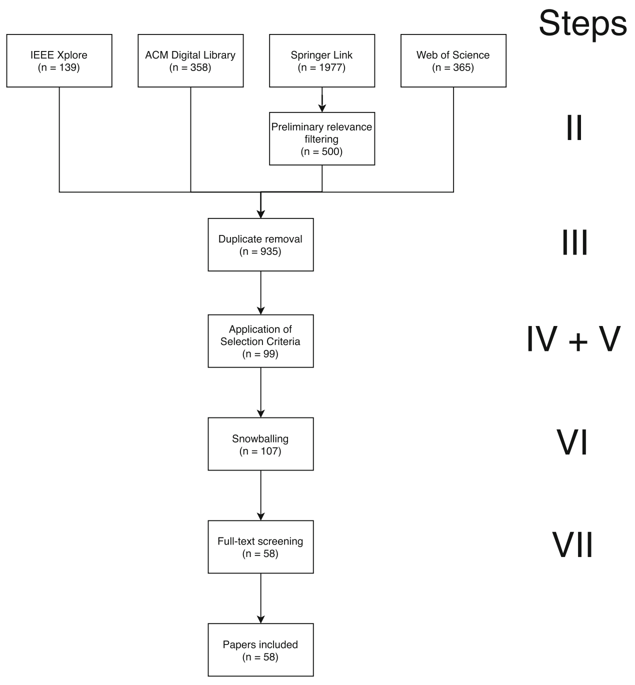
Steps
IEEE Xplore (n = 139)
ACM Digital Library (n = 358)
Springer Link (n = 1977)
Web of Science (n = 365)
Preliminary relevance filtering (n = 500)
II
Duplicate removal (n = 935)
III
Application of Selection Criteria (n = 99)
IV + V
Snowballing (n = 107)
VI
Full-text screening (n = 58)
VII
Papers included (n = 58)

Fig. 3 The search and selection process

This resulted in the search string shown in Fig. 2 which was applied in full text searches.

We decided on the following four search engines to use for automated literature search:

– ACM Digital Library

– IEEE Xplore
– Springer Link
– Web Of Science

Search engines were chosen based on their overall coverage, completeness, the availability of accessible publications and usage in other literature reviews in this field such as Loniewski, Insfran, and Abrahão [8,48]. The online library *Science Direct*, which is often used in this domain, was excluded from our list due to us only having limited access to the publications in the database. We decided that the overhead of requesting access to all publications for which our proceedings would require a full text review (see step 4) would take up too much time; thus, we excluded the database from our automatic search process. Badampudi, Wohlin, and Petersen [6] also show that combining the automatic database searches with an additional snowballing process can make up for a reduced list of searched databases. We also decided against using Google Scholar as a search engine due to our experience with it producing too many irrelevant results and having a large overlap with ACM Digital Library and IEEE.

We conducted several preliminary searches on all four databases during the construction of the search string, to validate the resulting publications included key publications.

After the definition and validation of the search string, the second step consisted of full text searches using the search engines of *ACM Digital Library*, *IEEE Xplore Digital Library* and *Web of Science*.

For the *Springer Link* database, we realized early on that a full text search would result in too many hits and instead opted to query only the titles for the keyword *model transformation language* and its synonyms and filtered these results by applying a full text search based on the remaining keywords and their synonyms. The remaining results still far exceeded those of all other databases combined. We further realized during preliminary sifting that neither title nor abstracts of publications beyond the first 200 results suggested a relevance to our study. For that reason, we decided to cap our search at 500 publications, doubling the size of results from the point where the relevance of publications started to slide. This decision is supported by the fact that any publication which ended up in our data extraction set was found within the first 200 results.

All automated database searches were conducted between June 17 and June 28, 2019.

In the third step, all duplicates that resulted from using multiple search engines were filtered out based on the publication title and date. This also included the removal of publications that had extended versions published in a journal. This resulted in a total of 935 publications.

During the fourth step, two researchers independently used the selection criteria (see Sect. 3.3) on the titles and abstracts to select a set of relevant publications. The researchers categorized literature as either *relevant* or *irrelevant*. And in cases where they could not deduce the relevance based on the title and abstract, the publication was marked as *undecidable*.

Afterwards, in step 5 the results for each publication of the independent selection processes were compared. In cases where the two researchers agreed on *relevant* or *irrelevant*, the paper was included or excluded from the final set of publications. In cases of either a disparity between the categorizations or an agreement on *undecidable*, the full text of the publications was consulted using adaptive reading techniques to decide whether it should be included or excluded. Adaptive reading in this context meant going from reading the introduction to reading the conclusion and if a decision was still not reached reading the paper from start to finish until a decision could be reached. The step resulted in a total of 99 publications to use as a start set for the sixth step.

In the sixth step, we applied exhaustive backward and forward snowballing, meaning, as described in many previous studies [5,59], until no new publication was selected. The snowballing procedures followed the guidelines laid out by Wohlin [67]. Our start set was comprised of all 99 publications from step 5. We then applied backward and forward snowballing to the set. For backward snowballing, we used the reference lists contained in the publications, and for forward snowballing we used Google Scholar as suggested by Wohlin [67] and because from our experience it provides the most reliable source for the cited by statistic. To the cited and citing publications, we then applied our inclusion and exclusion criteria as described in step 4. All publications that were deemed as relevant were then used as the starting set for the next round of snowballing until no new publications were selected as relevant. The result of this step was a set of 107 *relevant* publications.

Lastly, in step 7, we filtered out all publications that did not explicitly mention advantages or disadvantages of model transformation languages by reading the full text of all publications. This step was introduced to filter out the noise that arose from a broader search string and less restrictive inclusion criteria (see Sect. 3.3). The remaining 58 publications form our final set on which data synthesis was performed on. (A list of all included publications with an unique assigned ID can be found in "Appendix B".)

3.3 Selection criteria

We decided that a publication be marked as relevant, if it satisfies at least one inclusion criteria and does not satisfy any exclusion criteria. The inclusion criteria were chosen to include as many papers that potentially contain advantages or disadvantages as possible. A publication was included if:

IC1	The publication introduces a model transformation language.
IC2	The publication analyses or evaluates properties of one or multiple model transformation languages.
IC3	The publication describes the application of one or multiple model transformation languages.

IC1 is an inclusion criteria, because the introduction of a new language should include a motivation for the language and possibly even a section on potential shortcomings of the language. Such shortcomings can be attributed either to the design of the language or to the concept of model transformation languages as a whole.

A publication that is covered by *IC2* can help answer both *RQ1* and *RQ2* depending on the analysed/evaluated properties.

IC3 forms our third inclusion criteria since experience reports can be a good source for both strengths and weaknesses of any applied technique or tool.

Our exclusion criteria were:

EC1	Publications written in a language other than English.
EC2	Publications that are tutorial papers, poster papers or lecture slides.
EC3	Publications that are a Doctoral/Bachelor /Master thesis.

EC1 ensures that the scientific community is able to verify our extracted data from publications.

Because tutorial papers, poster papers and lecture slides are less reliable and do not provide enough information to work with, they are excluded with *EC2*.

Lastly, to reduce the required workload, we excluded all thesis publications with *EC3* as full text reviews would take up too much time. We also argue that relevant thesis findings are most likely also published in journal or conference papers.

3.4 Quality assessment checklist and procedures

Assessing the quality of publications found during the selection process is an essential part of a literature review [14].

For that reason, we adopted a list of six quality attributes for studies. The quality attributes (seen in Table 1) are taken from Shevtsov et al. [57] which adapted quality criteria from Weyns et al. [64]. Each quality item has a set of three characteristics for which a value between 0 and 2 is assigned. The quality score of a publication is calculated by summing up the values for each characteristic, making 12 the maximum quality score for a publication. The quality score did not influence the decision to include or exclude a publication.

3.5 Data extraction strategy

Based on our research questions, and general documentation concerns, we devised a total of eight data items to extract from each selected publication. Table 2 lists all extracted data items.

Data items *D1–D3* are recorded for documentation purposes.

To gather explicitly, claimed advantages and disadvantages of model transformation languages *D4* and *D5* are necessary items to include.

Another goal of our literature review is to find out which advantages or disadvantages are empirically verified. It is therefore necessary to extract information about whether empirical evidence exists and which advantage or disadvantage it is concerned with (*D6*). Similarly, citations used to back up claimed advantages or disadvantages are also documented (*D7*). Our goal is it to either track down references that provide evidence and find sources of common claims about advantages and disadvantages of model transformation languages.

Lastly, in order to evaluate the quality of publications the quality score *D8* for each publication is recorded.

All data items were extracted during full text reviews of all selected publications.

3.6 Synthesis procedures

The synthesis of the collected data was split into multiple parts with multiple results for each research question.

3.6.1 RQ1: What advantages and disadvantages of model transformation languages are claimed in the literature?

The first part of the synthesis for *RQ1* was a simple collection of all claimed advantages and disadvantages. This was done in order to create a basic overview.

Next, an analysis of all collected items was performed in order to devise categories for the advantages and disadvantages. To develop categories, we used initial coding and focused coding as described by Charmaz [19]. First, all claims were analysed claim by claim to extract common phrases or similar topics. These were then used to group together claims and develop descriptive terms when then served as the name for the category formed by the grouped claims. The categories themselves were split into a positive section and a negative section to contrast negative and positive mentions with each other.

Using the devised categorization allows for quick identification of contradictory claims. Such claims then have to be further analysed in terms of origin, context and supporting evidence.

Table 1 Quality assessment criteria [64]

Q1: Problem definition	
2	The authors provide an explicit problem description
1	The authors provide a general problem description
0	There is no problem description
Q2: Problem context	
2	If there is an explicit problem description for the research, this problem description is supported by references
1	If there is a general problem description, this problem description is supported by references
0	There is no description of the problem context
Q3: Research design	
2	The authors explicitly describe the plan (different steps, timing, etc.) they have used to perform the research, or the way the research was organized
1	The authors provide some general words about the research plan or the way the research was organized
0	There is no description of how the research was planned/organized
Q4: Contributions	
2	The authors explicitly list the contributions/results
1	The authors provide some general words about the results
0	There is no description of the research results
Q5: Insights	
2	The authors explicitly list insights/lessons learned
1	The authors provide some general words about insights/lessons learned
0	There is no description of the derived insights
Q6: Limitations	
2	The authors explicitly list problems and/or limitations
1	The authors provide some general words about limitations and/or problems
0	There is no description of the limitations

Table 2 Data items

ID	Data	Purpose
D1	Author(s)	Documentation
D2	Publication year	Documentation
D3	Title	Documentation
D4	Named advantage(s) of MTL(s)	RQ1
D5	Named disadvantage(s) MTL(s)	RQ1
D6	Empirical evidence of advantage(s) or disadvantage(s)	RQ2
D7	Cited evidence	RQ2
D8	Quality score	Documentation

3.6.2 RQ2: What advantages and disadvantages of model transformation languages are validated through empirical studies or by other means?

To analyse evidence of claimed advantages and disadvantage, we started by assessing the quality of each respective publication using the quality score system from Sect. 3.4.

Afterwards, we devised a visual representation for claims and evidence thereof in publications. The representation allows a straightforward identification of substantiated and unsubstantiated claims and tracking of citations back to the origin of cited claims. This in turn enabled us to easily identify whether citations back up stated claims or serve as nothing more than a reference to a publication which claims the same thing.

4 Findings

In this section, we provide a summary of the synthesized data as well as an analysis of the demographics and quality of publications. The summary will be in narrative form, supported by plots and graphs as suggested by Boot, Sutton and Papaioannou [14]. Before describing our findings with regard to the research questions from Sect. 3.1, we first offer statistics and information about the demographic data of the collected literature as well as an overview over their quality which we assessed using the quality criteria from Sect. 3.4.

4.1 Demographics

Figure 4 provides an overview over the quantity of included publications per year. An interesting thing to note is that it took only two years from the introduction of the *Model-Driven Architecture* in 2001 to the first mentions of advantages of model transformation languages. One of the most cited papers about model transformations in our literature review was published that year too (**P63**). Its title shapes introductions of publications in the community even today: *Model transformation: The heart and soul of model-driven software development*.

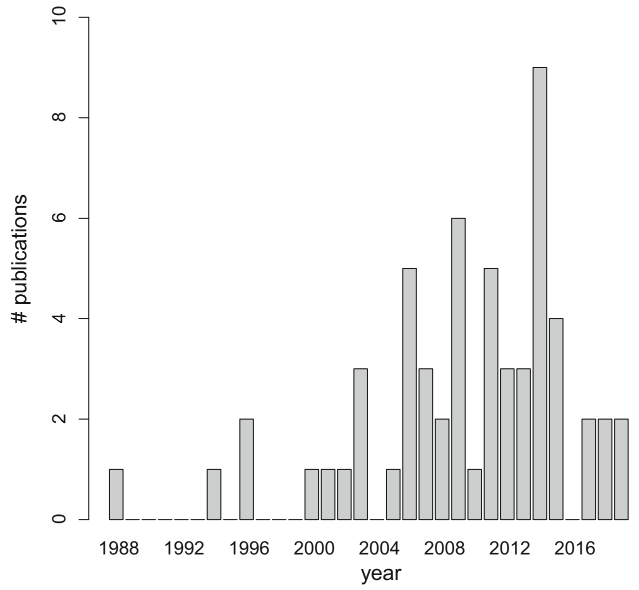

Fig. 4 Number of publications that mention or evaluate advantages or disadvantages of MTLs per year

Scrutinizing claims about MTLs, however, just recently started to be a focus of research, with the first study (**P59**) dedicated to evaluating advantages of MTLs being published in 2018. To us, this suggests that research might be slowly catching on to the fact that evaluation of specific properties of MTLs is necessary instead of relying on broad claims. Simply relying on the fact that model transformation languages are DSLs and that DSLs in general fare better compared to non-domain-specific languages [12,28,40] is not enough.

Industrial case studies about the adoption of MDSE have been performed much earlier than 2018, but such studies mainly focus on the complete MDSE workbench and do not analyse the impact of the used MTLs in great detail. The case study **P670** for example, while stating that "The technology used in the company should provide advanced features for developing and executing model transformations", does not go into detail about neither current shortcomings nor any

Table 3 Number of publications that mention specific MTLs

Model transformation language	# of mentions
ATL	16
EMT	1
ETL	3
GreAT	1
Henshin	1
Iquery	1
JTL	1
MOFLON	1
MT	1
NTL	2
QVT-O	4
QVT-R	2
SDM	1
SIGMA	1
SiTra	1
Tefkat	1
TGG	1
TN	1
VMTL	1

other specifics of model transformation languages used during the development process.

Overall, there are *32* publications that mention advantages and *36* publications that mention disadvantages. Moreover, *four* publications provide empirical evidence for either advantages or disadvantages, while *12* publications use citations to support their claims and *14* publications use other means such as examples and experience (more on this in Sect. 4.4).

Lastly, Table 3 shows which transformation languages were directly involved in publications used in our data extraction. We counted a transformation language as being involved if it was used, analysed or introduced in the publication. Simply being mentioned during enumerations of example MTLs was not sufficient.

The table paints an interesting picture. ATL far exceeds all other model transformation languages in involvement, and most languages are only discussed in a single publication.

4.2 Quality of publications

The results from the quality assessment, summarized in Fig. 5, shows that both the problem context and definition as well as the overall contributions are well defined in a majority of publications. Insights drawn from the work described in these publications, while less comprehensive in many cases, are also described most often. However, thorough descriptions of the research design, the used methods or steps taken

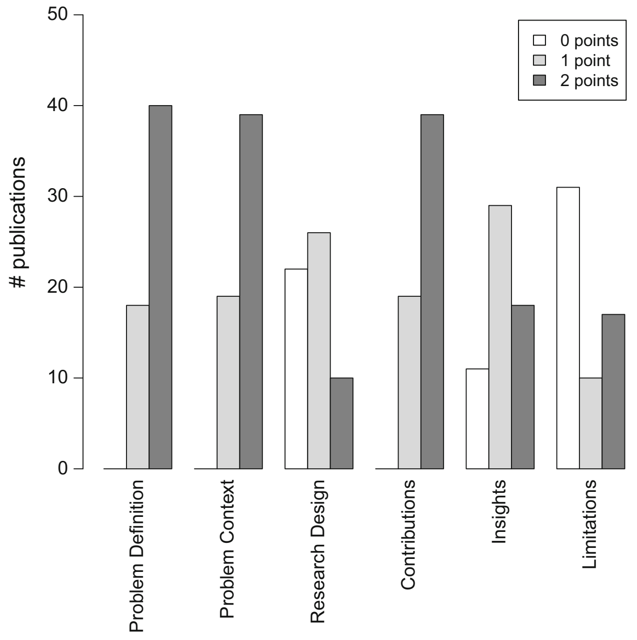

Fig. 5 Quality score distribution

are less common, a trend which is even more prominent for the presentation and discussion of limitations that act upon the studies. Similar observations have already been made by other literature reviews in different domains [26,57].

4.3 RQ1: Advantages and disadvantages of model transformation languages

We used data items D4 and D5 to answer our first research question, namely which advantages or disadvantages of dedicated model transformation languages are claimed in the literature. The resulting statements were sorted into 15 different categories (seen in Fig. 6) which arose naturally from the collected statements. An overview over all claims sorted into the different categories is given in Table 4. The table ascribes each claim with a unique ID (Cxx) for reference throughout this work. The table also contains evidence used to support a claim (if existent) to which we will come back later in Sect. 4.4. For almost all categories, there exist papers that describe model transformation languages as being advantageous as well as publications that describe them as disadvantageous in the category. In the following, we discuss the statements made in publications for each category.

4.3.1 Analysability

Throughout our gathered literature, there is only one publication, **P45**, that mentions analysability. According to them, a declarative transformation language comes with the added advantage of being automatically analysable which

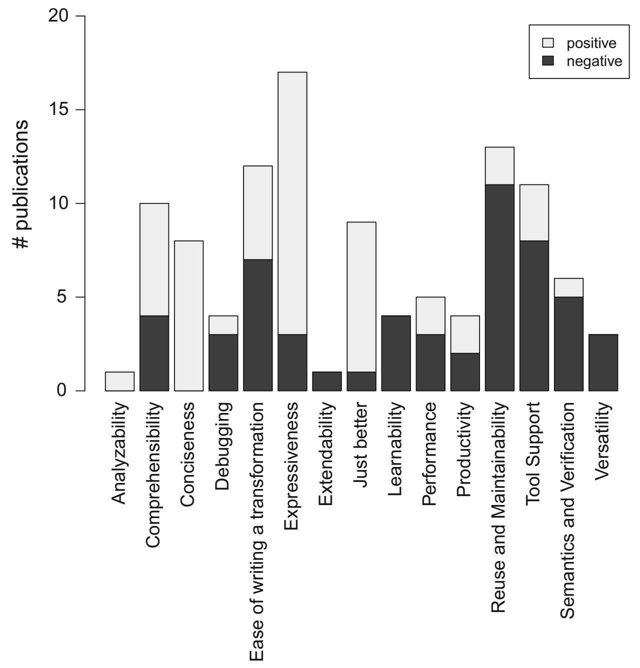

Fig. 6 Number of publications that claim an advantage or disadvantage of MTLs in a category

enables optimizations and specialized tool support (*C1*). While a detailed discussion of this claim within the publication remains owed, the authors provide examples of how static analysis allows the engine to implicitly construct an execution order. While our literature review found only a single publication that explicitly mentions analysability as an advantage of model transformation languages, there do exist multiple publications [2,3,63] that contain analysis procedures for model transformations.

4.3.2 Comprehensibility

Comprehensibility is a much disputed and multifaceted issue for model transformation languages. A total of eleven publications touch on several different aspects of how the use of MTLs influences the understandability of written transformations.

The first aspect is the use of graphical syntax compared to a textual one which is typically used in general-purpose programming languages. In **P63**, the authors talk about "perceived cognitive gains" of graphical representations of models when compared to textual ones (*C6*). A pronouncement that is echoed in **P43** states that graphical syntax for transformations is more intuitive and beneficial when reading transformation programs (*C2*).

While all these claims about graphical notation increasing the comprehensibility of transformations stand undisputed in our gathered literature, there are other facets in which graphical notation is said to be disadvantageous. We will come back to them later on in Sect. 4.3.5.

Declarative textual syntax is another commonly used syntax for defining model transformations. The authors of **P45** contend that a declarative syntax makes it easy to understand transformation rules in isolation and combination (*C3*). However, declarative transformation languages are typically based on graph transformation approaches which can become complex and hard to read according to **P70** (*C13*). They additionally assert that the use of abstract syntax hampers the comprehensibility of transformation rules (*C12*). Furthermore, **P22** insist that the use of graph patterns results in only parts of a meta-model being revealed in the transformation rules and that current transformation languages exhibit a general lack of facilities for understanding transformations (*C8*). **P22** also reports that understanding transformations in current model transformation languages is hampered, specially by the fact that many of the involved artefacts such as meta-models, models and transformation rules are scattered across multiple views (*C9*). **P29** brings forward the concern that large models are also a factor that hampers comprehensibility since there exist no language concepts to master this complexity (*C11*). Adding to this point, **P27** describes that for non-experts (e.g. stakeholders) transformations written in a traditional model transformation language are *"very complex to understand"* because they lack the necessary skills (*C10*). The authors of **P95** on the other hand claim that the usage of dedicated MTLs, which incorporate high-level abstractions, produces transformations that are more concise and more understandable (*C7*). This sentiment is shared in **P44** which explains the belief that using GPLs for defining synchronizations brings disadvantages in comprehensibility compared to model transformation languages (*C3*).

Understanding a transformation requires, among other things, understanding which elements are affected by it and in which context a transformation is placed. Using a model transformation language is beneficial for this as shown in the study described in **P59** (*C5*).

4.3.3 Conciseness

Interestingly, there seems to be a consensus on the conciseness of model transformation languages compared to GPLs.

In general, dedicated model transformation languages are seen as more concise (**P63** *C17*, **P95** *C21*) which, apart from textual languages, is also stated for graphical languages in **P75** (*C18*).

The fact that MTLs are more abstract making them more concise and thus better is claimed multiple times in **P80** (*C19*), **P52** (*C15*), **P3** (*C14*) and **P95** (*C20*), while **P673** claims that the abstraction in MTLs helps to reduce their overall complexity (*C22*).

The SLOC metric has also been drawn from as a way to compare MTLs with other MTLs and even GPLs. According to an experiment described in **P59**, using a rule-based model transformation reduces the transformation code by up to 48% (*C16*). Whether or not this is any indication of superiority is a disputed subject [9].

4.3.4 Debugging

Debugging support is much less disputed than comprehensibility. Of the five publications that talk about debugging in model transformation languages, none praise the current state of debugging support.

P22 (*C24*, *C25*) and **P90** (*C27*) both describe that currently no sufficient debugging support exist for MTLs. And while in **P95** it is stated that debugging of transformations in a dedicated languages is likely better than when the transformation is written in a general-purpose language (*C23*) they fail to bring forth a single example for their assertion.

Lastly, **P45** lauded declarative syntax for its benefit in comprehension but also note that imperative syntax is easier to debug in general (*C26*).

4.3.5 Ease of writing a transformation

The main purpose of model transformation languages is to improve the ease with which developers are able to define transformations. Hence, this should also be a main benefit when compared to general-purpose languages. However, the authors of the study described in **P59** found:*"no sufficient (statistically significant evidence) of general advantage of the specialized model transformation language QVTO over the modern GPL Xtend"* (*C39*). This is not to say that there are none as the authors admit the conclusions were *"made under narrow conditions"* but is still a concerning finding. Much more so because claims about such benefits of using MTLs persist through the literature. Claims such as those described in **P29** (*C29*), **P672** (*C32*) and **P50** (*C30*) state that their simpler syntax makes it easier to handle and transform models. These claims draw from statements about the expressiveness, to which we will come to in the next section, and reason that better expressiveness must lead to an easier time in writing transformations. A potential reason that hampers model transformation languages from evidentially being better for writing transformations is cited in **P27** (*C34*) and **P28** (*C35*). They both state that using a model transformation language requires skill, experience and a deep knowledge of the meta-models involved (**P56** *C38*). In our opinion, however, this holds true regardless of the language used to transform models.

Moreover, many model transformation languages use declarative syntax which can be unfamiliar for many programmers, according to **P45** (*C37*) and **P63** (*C40*), which are much more familiar with the status quo, i.e. imperative languages. The authors of **P22**, on the other hand, state that imperative MTLs often require additional code since many issues have to be accomplished explicitly compared to implicitly in declarative languages (*C33*).

Lastly, graphical syntax is said to make writing model transformations easier as the syntax is purported to be more intuitive for this task compared to a textual one in **P3**. In **P43** (*C36*) and **P672** (*C41*), however, the authors claim that graphical syntax can be complicated to use and that textual syntax is more compact and does not force users to spend time to beautify the layout of diagrams.

4.3.6 Expressiveness

As described in Sect. 2.2, the idea behind domain-specific languages is to design languages around a specific domain, thus making it more expressive for tasks within the domain [50]. Since model transformation languages are DSLs, it should not be a surprise that their expressiveness in the domain of model transformations is mentioned almost exclusively positive by a total of 19 different publications found in our literature review.

A large portion (**P95**, **P80**, **P94**, **P63**, **P15**, **P40**, **P52**, **P70**) of publications refer to expressiveness state that the higher level of abstraction that results from specific language constructs for model manipulation increases the conciseness and expressiveness of MTLs. **P80** additionally asserts that model transformation languages are just easier to use (*C61*).

Another portion (**P2**, **P15**, **P45**, **P677**, **P27**, **P63**, **P95**, **P27**) explains that the expressiveness is increased by the fact that model transformation engines can hide complexity from the developer. One such complex task is pattern matching and the source model traversal as mentioned in **P2** (*C42*), **P15** (*C43*) and **P45** (*C53*), respectively. According to them, not having to write the matching algorithms increases the expressiveness and ease of writing transformations in MTLs. Implicit rule ordering and rule triggering is another aspect that **P15** (*C46*), **P45** (*C51*) and **P677** (*C65*) claim increases the expressiveness of a transformation language. Related to rule ordering is the internal management and resolution of trace information which is stated by **P15** (*C44*), **P45** (*C50*), **P677** (*C65*) and **P95** (*C64*) to be a major advantage of model transformation languages. Furthermore, **P45** asserts that implicit target creation is another expressiveness advantage that MTLs can have over general-purpose languages (*C52*). Lastly, the study described in **P59** observed that copying complex structures can be done more effectively in MTLs (*C56*).

However, we also uncovered some shortcomings in current syntaxes. **P10** argues that the lack of expressions for transforming a single element into fragments of multiple targets is a detriment to the expressiveness of transformation languages, going as far as to allege that without such constructs model transformation languages are not expressive enough (*C68*). **P32** implies that MTLs are unable to transform OCL constraints on source model elements to target model elements (*C69*). And lastly **P33** critiques that model transformation languages lack mechanisms for describing and storing information about the properties of transformations (*C70*).

4.3.7 Extendability

Being able to extend the capabilities of a model transformation language seems to be less of a concern to the community. This can be seen by the fact that only **P50** touches this issue. They explain that external MTLs can only be extended (*"if at all"*) with a specific general-purpose language (*C71*). Internal model transformation languages of course do not suffer from this problem since they can be extended using the host language [21,32,46].

4.3.8 Just better

Apart from specific aspects in which the literature ascribes advantages or disadvantages to model transformation languages, there are also several instances where a much broader claim is made.

P86 for example states that there exists a consensus that MTLs are most suitable for defining model transformations (*C78*). This claim is also reiterated in several other publications using statements such as "the only sensible way" or "most potential due to being tailored to the purpose" (**P9**, **P23**, **P63**, **P64**, **P66**). However, one publication claims that both GPLs and MTLs are not well suited for model migrations and that instead dedicated migration languages are required (**P34** *C80*).

4.3.9 Learnability

The learnability issues of tools have been shown to positively correlate with usability defects [1] and thus their general acceptance.

However, the learnability of model transformation languages is rarely discussed in detail. **P30** (*C81*), **P58** (*C83*) and **P81** (*C84*) all express concerns about the steep learning curve of model transformation languages, and **P52** explain that transformation developers are often required to learn multiple languages, which requires both time and effort (*C82*).

4.3.10 Performance

The execution performance of transformations is an important aspect of model transformations. Often times, the goal is to trigger a chain of multiple transformations with each change to a model. Hence, good transformation performance is paramount to the success of model transformation languages.

Opinion on performance in the literature is divided. On the one hand, there are publications such as **P52** (*C88*) and **P80** (*C89*) which describe that the performance of dedicated MTLs is worse than that of compiled general-purpose programming languages, while on the other hand there is **P95** which states that some introduced transformation languages are more performant (*C85*), citing articles from the Transformation Tool Contest (TTC), and **P675** which shows a performance comparison of transformations written in Java and GrGen where GrGen performs better than Java (*C86*). There are also more nuanced views on the subject. **P45** describes that practitioners sometimes perceive the performance as worse and that there exist factors that hamper the performance (*C87*). The listed factors are the fact that the transformation languages are often interpreted, a mismatch with hardware and less control over the algorithms that are used. However, they also describe that specialized optimizations can bridge the performance gap.

4.3.11 Productivity

Increased productivity through the use of DSLs is a much cited advantage [50] (*C6D*). Unsurprisingly, it resurfaces in various forms in the context of model transformation languages as well. For instance, in **P45** it is described that the use of declarative MTLs improves the productivity of developers (*C91*). **P29** goes even further, claiming that the use of any model transformation language results in higher productivity (*C90*).

This is contrasted by the hypothesis that productivity in general-purpose programming languages might be higher due to the fact that it is easier to hire expert users, which was put forward in **P59** (*C93*). Lastly, **P32** raises the concern that some of the interviewed subjects perceive model transformation languages as not effective, i.e. not helpful for the productivity of developers (*C92*).

4.3.12 Reuse and maintainability

In our gathered literature, maintainability is used as a motivation for modularization and reuse concepts. **P29**,**P60** and **P95** all claim that reuse mechanisms are necessary to keep model transformations maintainable. Combined with a total of eight (**P4**, **P10**, **P29**, **P33**, **P41**, **P60**, **P95**, **P78**) publications that state that reuse is hardly, if at all, established in current model transformation languages, this paints a bleak picture for both maintainability and reuse. The need for reuse mechanisms has already been recognized in the research community as stated by **P77** in which the authors explain that a plethora of mechanisms have been introduced (*C95*) but are hindered by several barriers such as insufficient abstraction from meta-models and platform or missing repositories of reusable artefacts (*C103*).

There exists only a single claim that directly addresses maintainability. **P44** states that bidirectional model transformation languages have an advantage when it comes to maintenance (*C94*).

Apart from the maintainability of written code, there is also the maintainability of languages and their ecosystems. Surprisingly, this is hardly discussed in the literature at all. Only **P52** explains that evolving and maintaining a model transformation language is difficult and time-consuming (*C101*).

4.3.13 Semantics and verification

Three publications (**P39**, **P23**, **P58**) all suggest that most model transformation languages do not have well-defined semantics which in turn makes verification and verification support difficult (**P22** *C109*). **P44**, however, explains that bidirectional transformations are advantageous with regards to verification (*C107*).

4.3.14 Tool support

Tools are another important aspect in the MDE life cycle according to Hailpern and Tarr [28]. They are essential for efficient transformation development. Regrettably, MTLs lack good tool support according to **P23**, **P45**, **P52** and **P80** and if tools exist, they are not close to as mature as those of general-purpose languages as stated in **P74** (*C119*). Additionally, the authors of **P94** explain that developers of MTLs need to put extra effort into the creation of tool support for the language (*C121*). This might, however, be worthwhile, because **P44** presumes that dedicated tools for model transformation languages have the potential to be more powerful than tools for GPLs in the context of transformations (*C114*). And due to the high analysability of MTLs, **P45** explains that tool support could potentially thrive (*C115*). Internal MTLs, on the other hand, are able to inherit tool support from their host languages as reported by **P23** (*C113*). This helps to mitigate the overall lack of tool support, at least for internal MTLs.

An interesting discussion to be held is how important tool support for the acceptance of MTLs actually is. Whittle et al. [65] describe that organizational effects are far more impactful on the adoption of MDE, while the results of Cabot and Gérard [16] contradict this observation citing interviewees from commercial tool vendors that stopped the development of tools due to lack of customer interest.

4.3.15 Versatility

It should be self-evident that languages that are designed for a special purpose do not possess the same level of versatility and area of applicability than general-purpose languages. Hence, it is not surprising that all mentions of versatility of model transformation languages in our gathered literature paint MTLs as less versatile compared to GPLs (**P52** (*C124*), **P80** (*C125*), **P94** (*C127*)).

4.4 RQ2: Supporting evidence for advantages and disadvantages of MTLs

We found a number of different ways used by authors of our gathered literature to support their assertions. The largest portion of "supporting evidence" is made up of cited literature, i.e. a claim is followed by a citation that supposedly supports the claim.

The second way claims are supported is by example, i.e. authors implemented transformations in MTLs and/or GPLs and reported on their findings. Another aspect of this is relying on experience, i.e. authors state that from experience it is clear that some pronouncement is true or that it is a well-established fact within the community that a claim is true.

Third, there is empirical evidence, i.e. studies designed to measure specific effects of model transformation languages or case studies designed to gather the state of MTL usage in industry.

Last, there are those assertions that are not supported by any means. Authors simply suggest that an advantage or disadvantage exists. We assume that some claims made in this way implicitly rely on experience but do not state so. Nevertheless, since there is no way of testing this assumption we have to record such claims exactly the way they are made, without any evidence.

In the following sections, we will talk in detail about how each group of evidence is used in the literature to support claims about advantages or disadvantages of model transformation languages. As mentioned previously, Table 4 contains a complete overview over each claim and through what evidence the claim is supported.

4.4.1 Citation as evidence

Using citations to support statements is a core principle in research. It should therefore come as no surprise that citations are used to support claims about model transformation languages. An interesting aspect to explore for us was to trace how the cited literature supports the claim. For that, as stated in Sect. 3, we created a graphical representation to trace citations used as evidence through literature. The graph is shown in Fig. 7. It is inspired by UML syntax for object diagrams. The head of an "object" contains a *publication id*, while the body contains the categories for which advantages *(+)* or disadvantages *(–)* are claimed in the publication. Each category within the body is accompanied by an ID which can be used to find the corresponding claim within Table 4. We use different *borders* around publications to denote the type of evidence provided by the publication and *arrows* from one category within a publication to a different publication stand for the use of a citation to support a claim. Lastly, if the content of a publication does not concern itself with model transformation languages but instead with DSLs, the publication id is followed by "*(DSL)*".

Our graph allows to easily gauge information about the following things:

– What publication claims an advantage or disadvantage of MTLs in which category?
– What type of evidence (if any) is used to support claims in a publication?
– Which exact claims are supported through the citation of what publication?

In the following, we discuss observations about citations as evidence that can be made with help from the citation graphs.

First, only a total of 25 citations, split among 12 out of the 58 gathered publications, are used to support claims. This constitutes less than ten percent of all assertions found during our literature review. Seven of the 25 citations cite a publication that itself only states claims without any evidence thereof (**P63**, **P94**, **P673**, **P674**, **P800**). A further 11 end in a publication that uses examples or experience (see also Sect. 4.4.3) (**P664**, **P665**, **P667**, **P671**, **P672**, **P676**, **P77**, **P64**, **P804**, **P801**). Next, there are 3 citations that cite publications which in turn cite further publications to support their claims (**P677**, **P675**), leaving only 4 citations that cite empirical studies (**P669**, **P670**, **P803**) (see also Sect. 4.4.2). To us, this is worrying because the practice of citing literature that only restates an assertion corrodes the confidence readers can have in citations as supporting evidence.

From the graph, it is clearly evident that there exists no single cited source for claims about model transformation languages. This is clearly indicated by the fact that only five publications (**P63**, **P77**, **P673**, **P675**, **P803**) are cited more than once; twice to be exact. And no publication is cited more than two times. Moreover, of those five publications **P675** and **P803** are each cited by a single publication, respectively. **P675** is cited twice by **P80** and **P803** by **P675**. Related thereto, nearly each claim, even within the same category, is being supported through different citations.

Furthermore, only claims about *conciseness*, *expressiveness*, *reuse & maintainability*, *tool support*, *performance* and statements that MTLs are *just better* are supported using citations. It is interesting to note that claims within these categories which are supported by citations are either all positive or all negative. This is not to say that there are no contrasting claims, see for example *C113* and *C116* in **P23**, only that, if citations are used for a category the supported claims are either all positive or all negative.

Another thing to note is that in some instances claims about model transformation languages are being supported by citing publications on domain-specific languages in general. This can be seen in **P80**. The claims *C60* and *C61* are both supported by a citation of **P675** which is a publication that concerns itself with DSLs. Interestingly, **P675** itself then cites both publications about DSLs (**P800**, **P801**, **803**) and a publication about model transformation languages (**P804**) to support claims stated within the publication.

Coming back to citations of empirical studies, we have to report that while there exist 4 citations of empirical studies only a single claim about model transformation languages (*C116* in **P23**) is actually supported thereby. This is due to **P803** being an empirical study about DSLs and **P669** and **P670** both being cited as evidence for *C116*.

Lastly, apart from those publications that only make a single claim, no publication supports all their claims using citations. Extreme cases of this can be seen in **P45** and **P52** which make a total of 16 claims, only supporting three of them with citations while leaving the other 13 unsubstantiated.

4.4.2 Empirical evidence

To our disappointment, we have to report a lack of overall empirical evidence for properties of model transformation languages. Only four publications (**P32**, **P59**, **P669**, **P670**) in our gathered literature assess characteristics of model transformations using empirical means (see Fig. 7 and Table 4). Of those four, only **P59** focuses on MTLs as its central research object, while the other three are case studies about MDA that happen to contain results about transformation languages. **P803** too is an empirical study, but as mentioned in Sect. 4.4.1 focuses on domain-specific languages in general not on MTLs. In order to provide the necessary context for scrutinizing the claims extracted from the publications, we provide a short overview over the central aspects of **P32**, **P59**, **P669**, **P670** in the following.

The study described in **P59** was comprised of a large-scale controlled experiment with over 78 subjects from two universities as well as a preliminary study with a single individual. Subjects had to solve 231 tasks using three different languages (ATL, QVT-O and Xtend). The tasks focused on one of three aspects in transformation development, namely comprehending an existing transformation, changing a transformation and creating a transformation from scratch. After analysing the results, the authors come to the disillusioning conclusion that there is "no statistically significant benefit of using a dedicated transformation language over a modern general-purpose language".

The authors of **P32** report on an empirical study on the efficiency and effectiveness of MDA. A total of 38 subjects, selected from a model-driven engineering course, were asked

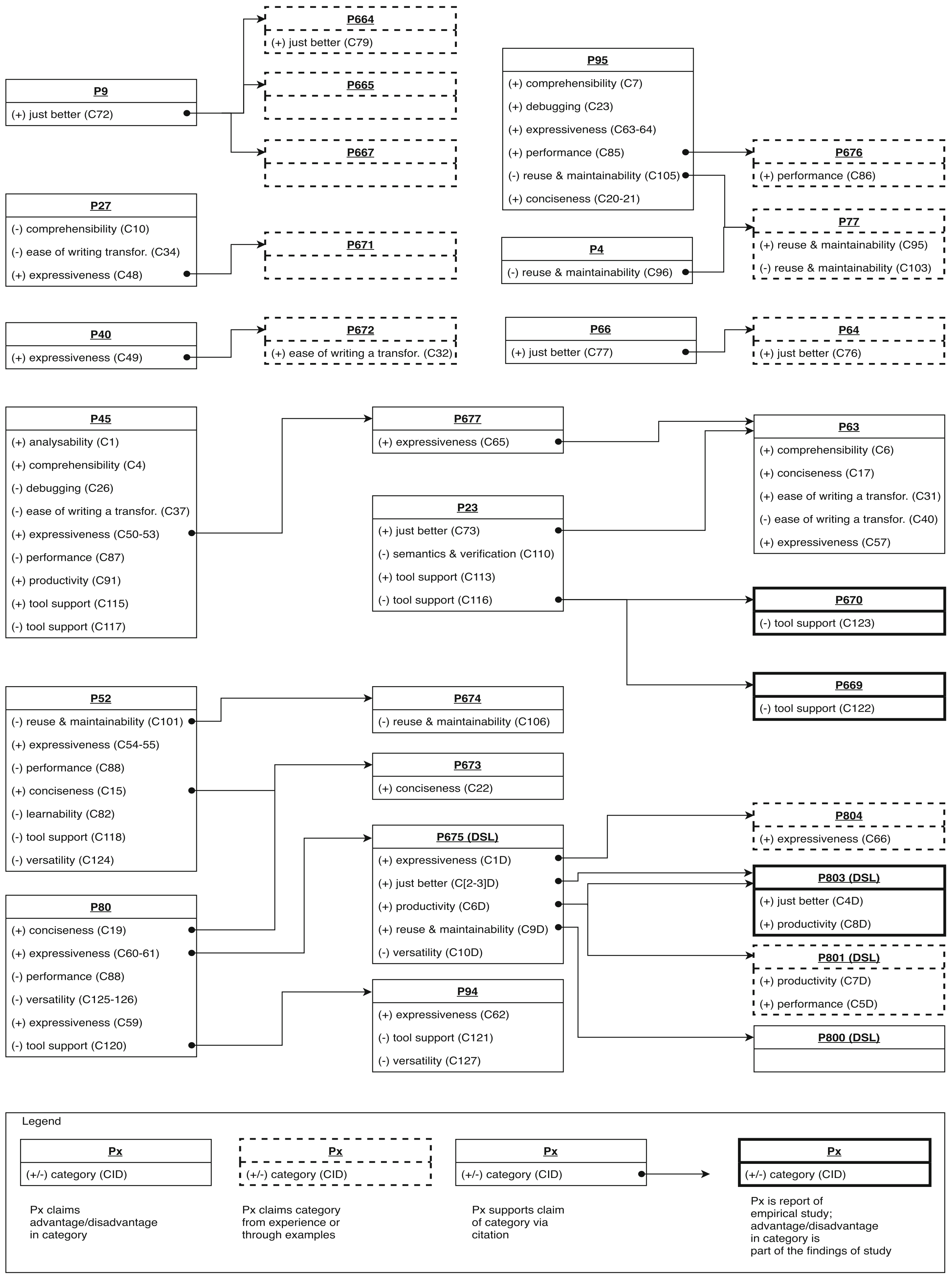

Fig. 7 Graph tracking citations of claims of various categories through literature

to implement the book-purchasing functionality of an e-book store system. Afterwards, the subjects evaluated the perceived efficiency and effectiveness of the used methodology. This also included questions about the used QVT language which was perceived as only marginally efficient.

Both **P669** and **670** are reports of industrial case studies. The objective of the study in **P669** was to investigate the state of practice of applying MDSE in industry. To achieve this, they collected data from tool evaluations, interviews and a survey. Four different companies were consulted to collect the data. Again while some reported results concerned themselves with transformations, model transformation languages were not explicitly discussed. Similarly, **P670** reports on an industrial case study involving two companies aiming to collect factors that influence the decision to adopt MDE. For that purpose, multiple preselected individuals at both companies were interviewed. Just as **P669**, the study did not directly focus on transformations or transformation languages.

As evident from Fig. 7, the results from **P32** and **P59** have yet to be used in the literature for supporting claims about MTLs. Since both of them have only been published recently, we are, however, optimistic about this prospect.

4.4.3 Evidence by example/experience

Using examples to demonstrate shortcomings of any kind has a long-standing tradition not only in informatics. Using examples to demonstrate an advantage, however, can result in less robust claims (especially toy or textbook examples Shaw [56]). As such, it is important to differentiate whether a claim is made by demonstrating a shortcoming or benefit.

In our gathered literature, ten publications use examples to support a claim. Interestingly, examples are mainly used to support broad claims about model transformation languages. This can be observed by the fact that **P34** and **P64** use examples to try and demonstrate that GPLs are not well suited for transforming models, while **P664**, **P665**, **P667**, **P672**, **P804** and **P676** try to demonstrate the general superiority of MTLs by showing examples of transformations written in MTLs. Other claims that are supported through examples are a demonstration of the reduction in code size when using rule-based MTLs in **P59** and statements about the extensive amount of reuse mechanisms for MTLs through listing gathered publications about the proposed mechanisms in **P77**.

Long-time practitioners of model transformation languages or programming languages in general often rely on their experience to make assertions about aspects of the language. And while the experience of long-term users can create valuable insights, it is still subjective and can therefore vary in accuracy. In our case, six publications directly state that their assertions come from experience. **P3** report on their experiences using different languages to implement transformations, coming to the conclusion that graphical rule definition is more intuitive, an experience shared by **P40**. **P43** name user feedback as grounds for claiming that visual syntax has advantages in comprehension but makes writing transformations more difficult. And **P672** share that they are under the impression that graph transformations are the superior method for defining refactorings.

Since experience is subjective, contradicting experiences are bound to occur sometime. While the authors of **P10** believe from experience that current MTLs are not abstract enough for expressing transformations, **P671** feel that the difficulty of writing transformations in a MTL does stem from the chosen MDD method rather than the syntax of the language.

4.4.4 No evidence

Figure 7 and especially Table 4 make it clear that a large portion of both positive and negative claims about model transformations are never substantiated. In fact, of the 127 claims ~69% are unsubstantiated. Adding those that are supported by a citation that in the end turns out to be unsupported as well brings the number up to ~77%. Particularly, the categories concerning the usability of MTLs such as *comprehensibility*, *ease of writing a transformation* and *productivity* lack meaningful evidence. All three of them being cornerstones of language engineers arguments for the superiority of model transformation languages make this especially worrisome.

We believe that a realization in the community about this fact is necessary. The necessity or superiority of model transformations has to be properly motivated. This means that it is not sufficient to claim advantages or disadvantages without providing at least some form of explanation on why this claim is valid (more on this in Sect. 5.3).

5 Discussion

In this section, we reflect on the previously presented findings. Our focus for this is fourfold. First, we feel it is necessary to draw parallels between our categorization and attributes of product quality. Next, we want to briefly discuss how claims are made in regards to transformation language features. Afterwards, a discussion about lack of empirical studies about properties of model transformation languages is warranted. And last we feel a discussion about the research direction for the community is also necessary.

5.1 Claims about model transformation languages in context of software quality

There are undeniable parallels between the categories we developed for claims and characteristics of software quality as defined by *ISO/IEC 25010:2011* [35]. This can be seen by the fact that many of our categories can be directly placed within the characteristics of the software product quality model (namely functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, portability).

Both *expressiveness* and *semantics and verification* are part of functional suitability. *Performance* and *productivity* can be classified under performance efficiency. Furthermore are *comprehensibility*, *conciseness*, *debugging*, *ease of writing a transformation*, *learnability* and *tool support* part of *Usability*. Maintainability covers *analysability* and *reuse & maintainability*. And lastly, *extendability* and *versatility* can be classified under portability. This leaves only our generic category *just better* without a corresponding characteristic which is to be expected.

However, there are also compatibility, reliability and security which have no corresponding categories from our categorization. This does not necessarily mean that the current research is not focused on aspects related to these quality criteria. It instead suggests a lack of concrete statements regarding them. And while security is justifiably less of a concern for model transformation languages, both the compatibility of different approaches and their reliability should definitely be focused on (see also Sect. 5.4).

Lastly, even though most claims we collected during our review could be categorized within the software product quality model we opted to develop a classification based on the claims alone since we believe the resulting categories to be more specialized and allow for a more nuanced view on the subject matter than the generic characteristics defined by *ISO/IEC 25010:2011* [35].

5.2 Claims about model transformation languages in context of language features

An effort by us to categorize the extracted claims along an existing taxonomy of model transformation language features such as the one by Czarnecki and Helsen [22] failed because a large portion of claims (~70%) are made broadly without reference to specific features of MTLs that aid the advantage or disadvantage.

We suggest that claims on benefits and disadvantages of model transformation languages be made more specific and include mentions of the features that aid or hamper the benefits. For example, incrementality aids the performance of model transformations since only parts of a transformation have to be re-executed and bidirectional transformation languages provide special support for incremental execution giving them an edge in performance.

5.3 Lack of evidence for MTL advantages and disadvantages

The current literature exhibits a deficit in evidence (empirical or otherwise) for asserted properties of model transformation languages. We believe there to be several factors which can explain this lack of evidence.

First, designing and conducting rigorous studies to examine model transformation languages requires a substantial amount of time and effort. Studies are further complicated by the lack of easily available study subjects due to the community being relatively small compared to the body of general-purpose programming language users. The study described in **P59**, for example, had to be conducted over the timespan of three semesters and at two universities just to attain 78 subjects. And even when a pertinent number of study subjects is found, ensuring comparable levels of experience within the subjects is another challenge, even more so when collaborating with industrial partners [58].

Relying on the fact that transformation languages are DSLs and hence bear all the benefits that are proclaimed for those might also be a factor. Describing the advantages of DSLs in the introduction of a paper about transformation languages is far from uncommon in the literature. And while we too believe that there are benefits when using DSLs, we would caution against broad usage of the fact that model transformation languages are DSLs to claim them advantageous over general-purpose languages (as is done in publications such as **P29**, **P63** or **P804**), especially because the manpower that goes into the development of the ecosystems of GPLs far exceeds that of MTLs.

Another problem is that statements can become "established" facts by virtue of being cited by a paper which is in turn cited. Suppose one author claims that model transformation languages are more expressive than GPLs. A second author claims the same thing and references the first author to provide context. Next, a third author, assuming that the second author verifies their claim via the citation, cites the second author to support a similar claim. Over time, this can lead to the statement being treated as a fact rather than an assumption made multiple times. This can be seen on multiple occasions in Fig. 7. **P63** makes an unsubstantiated claim (*C57*) that the expressiveness of MTLs is superior to that of GPLs. This claim is then reiterated by **P677** (*C65*) citing **P67**. Lastly, **P677** is cited by **P45** to support their assertion about the expressiveness of model transformation languages (*C50-53*). Such a chain is not even the worst case in our results. The chain **P80** → **P675** → **P801-804** is even more worrisome, in that some of the claims stated in **P80** (*C75*) actually originate in claims about domain-specific languages from **675** (*C1D*). **P80** claims two advantages of MTLs using **P675** as reference. **P675** again uses citations to support their claims. However, the papers cited by **P675** do not make statements about model transformation languages but DSLs in general. This shows how such chains can create a blurred factual picture. Moreover, in the presented cases it is still possible to find the origin of claims and realize how the claims were changed throughout the citation chains. If authors deemed it unnecessary to support claims that are "established" facts, this is no longer possible. Quite likely this is the case for a non-negligible number of publications (see Table 4) where no citations or any other substantiation for claimed properties of MTLs is given.

As previously described, it is not uncommon for authors to ascribe properties to model transformation languages due to them being DSLs. However, a language does not necessarily have to be more expressive, easier to use or easier to maintain simply by being domain specific. In fact we believe that everything about a DSL stands and falls with the domain itself as well as the design of the language. As a result, all advantages and disadvantages for DSLs, described in the literature, only define potential properties. Thus, it is necessary to evaluate advantages and disadvantages anew for each domain, especially in complex domains such as model transformations.

5.4 Research direction

In our opinion, the research community has to acknowledge that the current way of language development is not expedient. There needs to be a shift away from constant development of new features and transformation languages with, at best, prototypical evaluation. Tomaž Kosar, Bohra and Mernik [44] share this sentiment after a mapping study on the development of DSLs in general (see Sect. 6).

Instead, it is necessary to extensively evaluate current transformation languages, first to identify their actual strengths and weaknesses and then to compare these results with the expected (and desired) results to determine which aspects of MTLs still need improving.

We believe the categories from Sect. 4 to be a good reference for possible areas to evaluate.

It is not necessary to evaluate each category empirically: For some categories, empirical evaluation might not be sensible at all. Such categories include analysability, and semantics and verification for example, since there exist no universally accepted measures to base evaluation on. Additional literature reviews are also conceivable. Analogous to how **P77** gathered different reuse mechanisms, a comprehensive review of verification and analysis approaches can be useful to assess the *analysability* and *verifiability* of model transformation languages.

Designing and executing appropriate studies also entails significant effort which is why it becomes necessary to carefully weigh up which properties should be evaluated. Additionally, some categories should also be examined more urgently than others.

The *ease of writing a transformation* and *comprehensibility* are two such categories for which evaluation is most pressing. Also given that in the domain of programming languages (especially object-oriented programming), many studies exploring the comprehensibility and ease of use, such as Burkhardt et al. [15], Rein et al. [54], and Kurniawan and Xue [47], already exist. Study designs similar to the one described in **P59** are in our opinion most suitable for this purpose. This is supported by the fact that many studies for comparing programming languages follow a similar structure in that a common problem or task is solved in multiple languages and the resulting code is analysed [4,30,53]. It may also be useful to design the cases in such a way that the complete capabilities of the used transformation languages have to be used. In the study described in **P59**, for example advanced features such as QVTs *late resolve* were not part of the evaluation. Such a design can help to better understand if the most "advanced" features of transformation languages have practical value and how complex a GPL for these features is.

Comprehensibility can also be tested in isolation by requiring subjects to describe functionality of given transformations written in both a dedicated model transformation language and a GPL.

According to Mohagheghi et al. [51], one of the main motivations for adopting MDE in industry is to improve *productivity*; hence, we believe that evaluation of the productivity when using model transformation languages should be a focus too. Admittedly measuring productivity is a challenging task, a fact that has been observed as early as 1978 [37]. But since then, numerous ways have been proposed and tested out in practice [10,13] which should allow for productivity studies on MTLs to be carried out. A potential study into the productivity could require subjects to develop transformations in either a model transformation language or a general-purpose language within a certain time frame followed by measuring and comparing how productive the subjects were in both cases. Researchers can also draw from the large corpus of productivity studies on different aspects of programming, such as Wiger and Ab [66], Frakes and Succi [25] and Dieste et al. [23].

The *performance* of model transformations can have huge impact on development, especially when multiple transformations have to be executed in succession. Many language engineers already pay tribute to that fact by providing performance comparisons between their languages and other MTLs or general-purpose languages such as Java [32,46]. And the Transformation Tool Contest (TTC) provides a venue for comparing MTLs. However, we believe extensive comparisons between the performance of model transformation languages and general-purpose programming languages to be necessary to abolish the prejudice that dedicated transformation languages cannot outperform current compilers. Comparison of performance between different programming languages that are used for the same purpose is a well-established practice demonstrated by comparisons between Java and C++ for robotics programming done by Gherardi, Brugali and Comotti [27] or C++ and F90 for scientific programming by Cary et al. [18]. Performance comparisons are also common practice in other domains such as GPU programming where specialized DSLs are used and performance is of high importance (Karimi et al. [24]). It is conceivable to compare the performance of transformations written in dedicated MTLs and GPLs by either manually solving the same tasks as described previously or by using existing mechanisms (for example Calvar et al. [17]) for transforming transformation scripts written in a MTL into GPL code.

We also believe that special focus needs to be given to the question of what model transformation languages are expected to achieve (such as easy synchronization of multiple artefacts or fast transformations through incremental transformations): first, because this can allow to direct more resources on evaluating relevant aspects of MTLs; and second, because model transformation languages will appear more streamlined and mature when the focus of development lies in improving their core features instead of overloading them with “experimental” features. An opinion Tomaž Kosar et al. [44] share is that this can enable practitioners to truly understand the effectiveness and efficiencies of DSLs.

6 Related work

To the best of our knowledge, there exists no other literature review that explores advantages and disadvantages of model transformation languages. There does, however, exist some literature that can be related to our work.

A closely related survey and open discussion about the future of model transformation languages was held by Cabot and Gérard [16]. They report on the results of an online survey and subsequent open discussion during the 12th edition of the International Conference on Model Transformations (ICMT’2019). The survey was designed to gather information about why developers used MTLs or why they hesitate to do so and what their predictions about the future of these languages were. An open discussion was held after the results of the online survey were presented to the audience at ICMT’2019. The results of the study point towards MTLs becoming less popular not only because of technical issues but also due to tooling and social issues as well as the fact that some GPLs have assimilated ideas from MTLs and thus making them less bad alternatives to writing transformations in dedicated languages.

Hutchinson et al. [34] conducted an empirical study into MDSE in industry. The authors used questionnaires and interviews to explore different factors that influence the success of MDSE in organizations and attempt to provide empirical evidence for hailed benefits of MDSE. They report on a total of over 250 questionnaire responses as well as interviews with 22 practitioners from 17 different companies. While the main focus of the study was on MDSE adoption in general, the authors do report on some findings regarding model transformations, such as negative influences of writing and testing transformations on the productivity and influences of transformations on the portability. However, no results regarding used transformation languages are included.

Mens and Gorp [49] propose a taxonomy for model transformation languages. They define groups of transformation languages based on answers to a set of questions. The answers are split into multiple subgroups themselves. The authors describe in great detail different possible characteristics within the groups. In part, this also includes listings of properties for transformation languages that fall into specific groups. The authors, however, have not provided any evidence or more precise explanations. Similarly, Czarnecki and Helsen [22] propose a classification framework for model transformation approaches based on several approaches such as VIATRA, ATL and QVT. The framework is given as a feature diagram to allow to explicitly highlight different design choices for transformations. At the top level, the feature model contains features such as rule organization, incrementality, directionality and tracing. Each feature and its sub-components are extensively discussed and demonstrated with examples of transformation tools that boast different aspects of the features. In contrast to the two described classifications, our study categorizes claims about MTLs on a qualitative dimension rather than on language features.

Kahani et al. [39] describe a classification and comparison of a total of 60 model transformation tools. Their classification differentiates tools based on two levels. The first level describes whether the tool is a model-to-model (M2M) or model-to-text (M2T) tool. The second level differentiates M2M tools based on their transformation approach meaning whether the approach is relational, operational or graph-based and M2T tools based on the underlying implementation approach meaning visitor-based, template-based or hybrid. Unlike our study, the described comparison focuses on comparing different model transformation tools on a technical basis based on six categories (general, model level, transformation, user experience, collaboration support and runtime requirements), while we focus on qualitative aspects of claims made throughout literature about any kind of dedicated model transformation language.

Van Deursen et al. [62] gathered an annotated bibliography on the premise of *domain-specific languages versus generic programming languages*. The bibliography contains 73 different DSLs differentiated by their application domains: *Software Engineering*, *Systems Software*, *Multi-Media*, *Telecommunication* and *Miscellaneous*. Additionally, they provide a discussion of terminology as well as risks and benefits of DSLs. And while parts of the listed risks and benefits such as enhanced productivity or cost of education can be found in the listed advantages and disadvantages of our literature review, their bibliography does not contain any model transformation languages.

Tomaž Kosar et al. [44] report on the results of a systematic mapping study they conducted to understand the DSL research field, to identify research trends and to detect open issues. Their data comprised a total of 1153 candidates which they condensed into 390 publications for classification. The results from the study corroborate observations made during our literature review. The research community is mainly concerned with the development of new techniques, while research into the effectiveness of languages and empirical evaluations is lacking.

Tomaz Kosar et al. [45] describe an empirical study comparing a domain-specific language with a general-purpose language with a focus on learning, perceiving and evolving programs. The two languages considered were XAML as a DSL representative and the GPL C#. The experiment is comprised of 36 programmers which were asked to construct a graphical interface using both XAML and C# Forms. Afterwards, the subjects had to answer a questionnaire. In contrast to the results of **P59**, their results show a statistically significant advantage of DSLs for learning, comprehending and evolving programs.

Jakumeit et al. [36] provide an extensive overview over and comparison of 13 state-of-the-art transformation tools used in the TTC 2011. The authors give detailed descriptions of the tools based on a "Hello World" case posed at the contest. They also describe for what use cases the individual tools are best suited and provide a novel taxonomy based on which the tools are compared. The introduced taxonomy features many of the same categories we synthesized from the claims in our literature review, such as expressiveness, extendability, learnability, reuse and verification, but also other categories such as maturity and license.

7 Threats to validity

To ensure reproducibility and a high quality of the results, we followed a systematic approach as detailed in Sect. 3. However, possible threats to validity still remain. In this section we discuss these threats.

7.1 Internal validity

Internal validity describes the extent to which a casual conclusion based on the study is warranted. This validity is threatened by possible differences in the interpretation of our selection criteria. To alleviate the potential threat, two researchers independently applied the selection criteria and in cases of different decisions about the inclusion of a publication, full text cross-reading was applied.

A threat to the internal validity we could not meet with prevention measures was the fact that our categorization is based on certain defining expressions like *"expressive"* and *"versatile"*. It is possible that different authors ascribe different meanings to these phrases. While we believe that for most cases this is less of a problem, it is still a problem that we could not fully solve since not every publication defines their understanding of used phrases.

7.2 External validity

External validity describes the extent to which the findings of a study can be generalized. For structured literature reviews, a threat to this validity arises from the existence of relevant but undetected or excluded publications [20]. To mitigate this threat as much as possible, we used both automatic searches and exhaustive backward and forward snowballing. The automatic search was also conducted on multiple literature databases to broaden the field of searched literature. Furthermore, we employed a "when uncertain include" strategy for including publications, as well as less strict inclusion criteria which helped prevent relevant publications from being overlooked.

7.3 Construct validity

Construct validity describes the extent to which the right measures were obtained and whether the right scope was defined in relation to our research questions. The construct validity of our research is not under threat since the research questions define easily producible results. Cited advantages or disadvantages of model transformation languages can be directly extracted, and the same also holds for used evidence for claims.

7.4 Conclusion validity

Conclusion validity describes the extent to which conclusions based on data are reproducible.

Prior to the execution of our literature review, we defined a review protocol for all phases of the review. We followed the protocol rigorously to ensure reproducibility of the study. The protocol did not only include descriptions of how the review had to be conducted but also detailed how data should be extracted from the selected literature (see Sect. 3). It is of course possible that, with the passage of time, a repetition of the literature review can draw different conclusions due to the added body of literature between then and now.

8 Conclusion

In this study, we have reported on a systematic literature review intended to extract and categorize claims about model transformation languages as well as the current state of evaluation thereof. The goal of the study was to compile a comprehensive list and the categorization of positive and negative claims about model transformation languages. We further wanted to investigate the current state of evaluation of claims as well as identify gaps in the area of evaluation of MTLs.

We combed over 4000 publications for that purpose, 58 of which we selected for the study. To this end, we followed a rigorous process by using a combination of automatic searches on literature databases, exhaustive backward and forward snowballing and multiple researchers during the selection phase. The selected publications were combed for mentions of advantages and disadvantages of MTLs and evidence of the stated claims. Lastly, we analysed and discussed the extracted claims and evidence to: (i) provide an overview over claimed advantages and disadvantages and their origin, (ii) the current state of evidence thereof and (iii) identify areas where further research is necessary.

We conclude that: (i) the current literature claims many advantages of MTLs but also points towards deficits owed to the mostly experimental nature of the languages and its limited domain, (ii) there is insufficient evidence for and (iii) research about properties of model transformation languages.

The results of our study suggest that there is much to be done in terms of evaluation of model transformation languages and that effort that is currently being invested into the development of new features might be better spent evaluating the state of the art in hopes of ascertaining both what current MTLs are lacking most and where their strengths really lie.

Acknowledgements Open Access funding provided by Projekt DEAL.

A Overview over all extracted claims

See Table 4.

Table 4 Overview over claims per category

Category	Valuation	CID	Claim	Publication	Evidence
Analysability	Positive	C1	Declarative MTLs lend themselves to automatic analysis	**P45**	–
Comprehensibility	Positive	C2	Based on user feedback, it was identified that visual syntax is beneficial when reading a transformation program	**P43**	Experience
		C3	Bidirectional transformation languages have an advantage in comprehensibility	**P44**	–
		C4	Rules written in a declarative MTL are more easily understood in isolation and in combination	**P45**	–
		C5	An observation made from the empirical data is that context selection and identification are easier for subjects working with MTLs than with GPLs	**P59**	Empirical study

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
		C6	There are perceived cognitive gains of graphical representations compared to fully textual representations of transformations shown by for example the appeal of UMLs graphical representation of models	**P63**	–
		C7	Model transformation languages incorporate high-level abstractions that make them more understandable than GPLs	**P95**	–
	Negative	C8	Comprehensibility of transformation logic is hampered as current transformation languages provide only a limited view on a transformation problem. For example, graph transformation approaches only reveal parts of the meta-model	**P22**	–
		C9	Most MTLs lack convenient facilities for understanding the transformation logic	**P22**	–
		C10	Model transformation languages require specific skills and as a result are hard to understand for many stakeholders	**P27**	–
		C11	Large and heterogeneous models lead to poorly understandable transformation code due to missing language concepts to master complexity	**P29**	–
		C12	Graph transformations defined on abstract syntax are hard to read because the user has to be familiar with meta-model that defines the abstract syntax	**P70**	–
		C13	Purely graph-based transformation languages can become complex and hard to read	**P70**	–
Conciseness	Positive	C14	General-purpose languages lack simplicity because of how transformations are defined	**P3**	Examples
		C15	GPLs do not allow developers to conveniently express model manipulation concepts and the loss of abstraction in GPLs may give rise to accidental complexities	**P52**	Cites **P673**
		C16	Transformations implemented in the pre-study using rule-based MTLs were up to 48% smaller than corresponding Java variants	**P59**	Preliminary study
		C17	Declarative approaches make language more concise	**P63**	–
		C18	Graphical notation in MTLs is concise	**P75**	–

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
		C19	GPLs do not conveniently express model manipulation concepts and the loss of abstraction can give rise to accidental complexities	**P80**	Cites **673**
		C20	Model transformation languages incorporate high-level abstractions that make them more concise than GPLs	**P95**	–
		C21	Model transformation languages are more concise	**P95**	–
		C22	MDE and model transformation languages such as QVT help to reduce complexity	**P673**	–
Debugging	Positive	C23	Debuggers for MTLs are likely better than those for GPLs for debugging transformations since it is questionable whether the call stacks produced by debuggers of GPLs are meaningful for the developer	**P95**	–
	Negative	C24	Although numerous transformation languages exist, they lack convenient facilities for supporting debugging and understanding of the transformation logic	**P22**	–
		C25	In ATL, TGGs and QVT-R correspondence is defined on a higher level of abstraction compared to on what execution engines operate Thus debugging is limited on the lower level of the execution engines not on the level of the language definition	**P22**	–
		C26	In declarative model transformation languages debugging is more difficult than in imperative ones	**P45**	–
		C27	Model transformation languages lack proper debugging support since implementation cost is high	**P90**	–
Ease of writing a transformation	Positive	C28	We found graphical rule definition far more intuitive than syntax-based definition	**P3**	Experience
		C29	Model transformation languages ease development efforts by offering succinct syntax to query from and map model elements between different modelling domains	**P29**	–
		C30	Model transformation languages make it easy to work with models	**P50**	–
		C31	Imperative transformation approaches offer a familiar paradigm, that is, sequence, selection, and iteration	**P63**	–

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
		C32	It is our impression that, in general, graph transformations offer significantly better support for the specification and implementation of modelling guidelines and refactorings	**P672**	Experience
	Negative	C33	Imperative MTLs induce overhead code because many issues have to be accomplished explicitly, e.g. specification of control flow	**P22**	–
		C34	Traditional transformation languages require specific skills to be able to write transformations	**P27**	–
		C35	To be able to write transformations, one has to be a transformation expert	**P28**	–
		C36	Based on user feedback we identified that writing a transformation program with a graphical syntax can be complicated	**P43**	Experience
		C37	The syntax of declarative MTLs is unfamiliar for many developers	**P45**	–
		C38	Model transformation languages that define transformations on meta-model level require deep understanding of the meta-model	**P56**	–
		C39	There is no sufficient (statistically significant) evidence of a general advantage of specialized model transformation languages (ATL, QVT-O) over a modern GPL (Xtend)	**P59**	Empirical study
		C40	Developers are generally more comfortable with encoding complicated (transformation) algorithms in procedural languages	**P63**	–
		C41	First of all, some of us are not convinced that the usage of a visual notation has significant advantages compared to a textual notation. A textual notation is more compact, simplifies all kinds of version and configuration management tasks, and does not force its users to spend hours beautifying the layout of huge diagrams	**P672**	–
Expressiveness	Positive	C42	Rule-based approaches seem to be less error-prone compared to a manual implementation of pattern matching for each transformation in a general-purpose language	**P2**	–
		C43	Model transformation languages can hide details like traversing behind simple syntax	**P15**	–

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
		C44	Model transformation languages can hide traces behind simple syntax	**P15**	–
		C45	Model transformation languages can hide rule triggering behind simple syntax	**P15**	–
		C46	Model transformation languages can hide rule ordering behind simple syntax	**P15**	–
		C47	Model transformation languages can hide complex transformation algorithms behind a simple syntax	**P15**	–
		C48	Model transformation languages hide transformation complexity and burden from user	**P27**	Cites **P671**
		C49	Graph transformations generally offer a significantly better support for the specification and implementation of modelling guidelines and refactorings	**P40**	Cites **P672**
		C50	Declarative MTLs allow automatic traceability management	**P45**	Cites **P677**
		C51	Declarative model transformation languages allow for implicit rule ordering lessening the load on developer	**P45**	Cites **P677**
		C52	Declarative MTLs can do implicit target object creation	**P45**	Cites **P677**
		C53	Declarative MTLs allow for implicit source model traversal	**P45**	Cites **P677**
		C54	Model transformation languages syntax is more specific	**P52**	–
		C55	GPLs do not allow developers to conveniently express model manipulation concepts	**P52**	–
		C56	We found that copying complex structures is more effective in MTLs	**P59**	Empirical study
Expressiveness	Positive	C57	General-purpose languages lack suitable abstractions for specifying transformations	**P63**	–
		C58	Graph-based MTLs are especially popular due to their high expressive power	**P70**	–
		C59	Model transformation languages have more specific language constructs	**P80**	–
		C60	Model transformation languages have a higher level of abstraction which leads to gains in expressiveness over GPLs	**P80**	Cites **P675**
		C61	Model transformation languages are easier to use than GPLs	**P80**	Cites **P675**

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
		C62	Model transformation languages transformation constructs are more specific	**P94**	–
		C63	From our perspective, automatic handling/resolution of traces by transformation engine is one of the major features that make existing MTLs better suited for model transformations than GPLs	**P95**	–
		C64	General-purpose languages lack sufficient transformation concepts	**P95**	–
		C1D	DSLs trade expressiveness in a limited domain for generality	**P675**	Cites **P804**
		C65	GPLs lack suitable abstractions for specifying transformations	**P677**	Cites **P63**
		C66	With a DSL/MTL, a programmer can express their objective in a concise manner using a language that is much higher in expressiveness than that typically offered in a transitional programming language	**P804**	–
	Negative	C67	Having written several transformation, we have identified that current MTLs are too low a level of abstraction for succinctly expressing transformations between DSLs because they demonstrate several recurring patterns that have to be reimplemented each time	**P10**	Experience
		C68	Having written several transformation, we have identified that mapping a single element to fragments of multiple elements has to be done programmatically which is counterintuitive and error-prone	**P10**	Experience
		C69	OCL constraints cannot be transformed in MTLs	**P32**	Empirical Study
		C70	There is no mechanism for describing and/or storing information about the properties of a transformation	**P33**	–
Extendability	Negative	C71	Extending model transformation languages is difficult	**P50**	–
Just better	Positive	C72	GGT (graph grammar and graph transformation) are a powerful technique for specifying complex transformations	**P9**	Cites **P664-P666**
		C73	General-purpose programming languages are not suitable for defining model transformations	**P23**	Cites **P63**

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
		C74	GPLs are not well suited for model migration	**P34**	Examples
		C75	Dedicated MTLs offer the most potential transformation approach because the languages can be tailored for the purpose	**P63**	–
		C76	In order to transform models in a GPL, one has to add increasing amounts of machinery, e.g. to keep track of which elements have already been transformed. This leads to the assumption that model transformations cannot be sensibly written in a standard programming language	**P64**	Examples
		C77	Model transformations present a number of problems which imply that dedicated approaches are required	**P66**	Cites **P64**
		C78	The current consensus is that specialized languages with a mixture of declarative and imperative constructs are most suitable for specifying model transformations	**P86**	–
		C79	With the help of an example, we have shown that GGT (graph grammar and graph transformation) can be used to transform PIMs into PSMs	**P664**	Examples
		C2D	DSLs open up the application domain to a larger group of developers	**P675**	Cites **P803**
		C3D	Domain-specific languages increase the ease of use	**P675**	Cites **P803**
		C4D	When using DSLs, less errors are made	**P803**	Empirical Study
	Negative	C80	General-purpose MTLs are not well suited for model migration since there is additional overhead but dedicated migration languages are	**P34**	Examples
Learnability	Negative	C81	The generality of general-purpose MTLs can have the effect of making them less approachable and create a steep learning curve for non-expert users	**P30**	–
		C82	Users have to learn multiple similar, but not always consistent, languages, which requires considerable time to learn	**P52**	–
		C83	Model transformation languages have a steep learning curve	**P58**	–
		C84	One has to learn a completely new language to transform models with MTLs	**P81**	–
Performance	Positive	C85	Model transformation languages are more performant	**P95**	Cites **P676**
		C86	GrGen shows a better performance of transformations than Java	**P676**	Samples
		C5D	DSLs have better performance	**P801**	–

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
	Negative	C87	Declarative MTLs have performance problems	**P45**	–
		C88	The performance of model transformation languages is a shortcoming that may make users feel limited	**P52**	–
		C89	MTLs have worse performance	**P80**	–
Productivity	Positive	C90	Model transformation languages being DSLs improve the productivity	**P29**	–
		C91	Declarative MTLs increase programmer productivity	**P45**	–
		C6D	DSLs increase productivity	**P675**	Cites **P801**, **P803**
		C7D	Using DSLs increases productivity	**P801**	Examples
		C8D	Using DSLs increases productivity	**P803**	Empirical Study
	Negative	C92	The perceived effectiveness of model transformation languages is bad	**P32**	Empirical Study
		C93	Productivity of GPL development might be higher since expert users for GPLs are easier to hire	**P59**	–
Reuse and Maintainability	Positive	C94	Bidirectional model transformations have an advantage in maintainability	**P44**	–
		C95	There exists a plethora of reuse mechanisms for MTLs	**P77**	Literature review
		C9D	Domain-specific languages reduce the maintenance costs	**P675**	Cites **P800**
	Negative	C96	Reuse is sparse, transformations are written from scratch every time because meta-models differ slightly	**P4**	Cites **P77**
		C97	Having written several transformation, we have identified that recurring patterns have to be implemented from scratch every time	**P10**	Experience
		C98	There exists no module concept for model transformation languages that allows programmers to control information hiding and strictly declare model and code dependencies at module interface	**P29**	–
		C99	Model transformation languages lack sophisticated reuse mechanisms	**P33**	–
		C100	Unfortunately, the definition of model transformations is normally a type-centric activity, thus making their reuse for other meta-models difficult	**P41**	–
		C101	Evolving and maintaining MTL requires effort	**P52**	Cites **P674**

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
		C102	The emphasis of MDE on using DSLs has caused a proliferation of meta-models. In this scenario, developing a transformation for a new meta-model is usually performed manually with no reuse, even if comparable transformations for similar meta-models exist	**P60**	–
		C103	There are barriers such as insufficient abstraction of reuse mechanisms from meta-models that hamper reuse	**P77**	Literature review
		C104	There is little support for reusing model transformations in different contexts since they are tightly coupled to the meta-models they are defined upon	**P78**	–
		C105	Reuse of model transformations is hardly established in practice	**P95**	Cites **P77**
		C106	Developing these new languages to a sufficient degree of maturity is an enormous effort which includes for example construction and optimisation of compilers	**P674**	–
Semantics and Verification	Positive	C107	Bidirectional transformation languages have an advantage in verification	**P44**	–
	Negative	C108	For existing relational model transformation approaches, it is usually not really clear under which constraints particular implementations really conform to the formal semantics	**P21**	–
		C109	Comprehensive verification support of model transformations is missing	**P22**	–
		C110	There is a semantic difference between a typical programming language and formalisms that support bidirectionality and change propagation such as TGGs	**P23**	–
		C111	Most transformation languages have no formal semantics to add detailed specifications on the expected behaviour	**P39**	–
		C112	The semantics of many model transformation languages is not formally defined	**P58**	–

Table 4 continued

Category	Valuation	CID	Claim	Publication	Evidence
Tool Support	Positive	C113	Internal MTLs can inherit tool support of general-purpose host language	**P23**	–
		C114	Tool support for external transformation languages is potentially more powerful than for internal MTL or GPL because it can be tailored to the DSL	**P44**	–
		C115	Declarative MTLs provide opportunities for specialized tool support	**P45**	–
	Negative	C116	Model transformation languages lack tool support	**P23**	Cites **P669**,**P670**
		C117	Declarative MTLs lack libraries and tool support	**P45**	–
		C118	Model transformation languages lack tool support	**P52**	–
		C119	Supporting tools for MTLs have not the same level of maturity as for GPLs	**P74**	–
		C120	Model transformation languages have worse tool support	**P80**	Cites **P94**
		C121	Tool support for external MTLs has to be developed which entails extra effort	**P94**	–
		C122	Tool support for model transformations is not as mature as subjects would like	**P669**	Empirical study
		C123	Tool support for model transformations is not great	**P670**	Empirical study
Versatility	Negative	C124	The syntax of model transformation languages is less versatile	**P52**	–
		C125	Model transformation languages are less versatile than GPLs	**P80**	–
		C126	Model transformation languages have less versatile language constructs	**P80**	–
		C127	Model transformation languages constructs are less versatile	**P94**	–
		C10D	DSLs are less general than general-purpose programming languages	**P675**	–

B SLR results

P2 Patzina, Sven and Lars Patzina (2012). "A Case Study Based Comparison of ATL and SDM". In: *Proceedings of the 4th International Conference on Applications of Graph Transformations with Industrial Relevance*. AGTIVE 2011. DOI: https://doi.org/10.1007/978-3-642-34176-2_18.

P3 Stephan, Matthew and Andrew Stevenson (2009). *A Comparative Look at Model Transformation Languages*. Tech. rep. Software Technology Laboratory at Queens University. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.712.2983.

P4 Cuadrado, J. S., E. Guerra, and J. de Lara (2014). "A Component Model for Model Transformations". In: *IEEE Transactions on Software Engineering*. DOI: https://doi.org/10.1109/TSE.2014.2339852.

P9 Agrawal, Aditya, Gabor Karsai, and Feng Shi (2003). "A UML-based graph transformation approach for implementing domain-specific model transformations". In: *International Journal on Software and Systems Modeling*. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.152.1226.

P10 Johannes, Jendrik et al. (2009). "Abstracting Complex Languages through Transformation and Composition". In: *Model Driven Engineering Languages and Systems*. MODELS 2009. DOI: https://doi.org/10.1007/978-3-642-04425-0_41.

P15 Jouault, Frédéric et al. (2008). "ATL: A model transformation tool". In: *Science of Computer Programming*. DOI: https://doi.org/10.1016/j.scico.2007.08.002.

P21 Giese, Holger, Stephan Hildebrandt, and Leen Lambers (2014). "Bridging the gap between formal semantics and implementation of triple graph grammars". In: *Software & Systems Modeling*. DOI: https://doi.org/10.1007/s10270-012-0247-y.

P22 Schoenboeck, Johannes et al. (2010). "Catch Me If You Can - Debugging Support for Model Transformations". In: *Models in Software Engineering*. MODELS 2009. DOI: https://doi.org/10.1007/978-3-642-12261-3_2.

P23 Hinkel, Georg and Erik Burger (2019). "Change propagation and bidirectionality in internal transformation DSLs". In: *Software & Systems Modeling*. DOI: https://doi.org/10.1007/s10270-017-0617-6.

P27 Sottet, J. and A. Vagner (2014). "Defining Domain Specific Transformations in Human-Computer interfaces development". In: *2014 2nd International Conference on Model-Driven Engineering and Software Development*. MODELSWARD '14. URL: https://ieeexplore.ieee.org/abstract/document/7018471.

P28 Acretoaie, Vlad (2013). *Delivering the Next Generation of Model Transformation Languages and Tools*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.708.6612.

P29 Rentschler, Andreas et al. (2014). "Designing Information Hiding Modularity for Model Transformation Languages". In: *Proceedings of the 13th International Conference on Modularity*. MODULARITY '14. DOI: https://doi.org/10.1145/2577080.2577094.

P30 Steel, Jim and Robin Drogemuller (2011). "Domain-Specific Model Transformation in Building Quantity Take-Off". In: *Model Driven Engineering Languages and Systems*. MODELS 2011. DOI: https://doi.org/10.1007/978-3-642-24485-8_15.

P32 Shin, Shin-Shing (2019). "Empirical study on the effectiveness and efficiency of model-driven architecture techniques". In: *Software & Systems Modeling*. DOI: https://doi.org/10.1007/s10270-018-00711-y.

P33 Criado, Javier et al. (2015). "Enabling the Reuse of Stored Model Transformations Through Annotations". In: *Theory and Practice of Model Transformations*. ICMT 2015. DOI: https://doi.org/10.1007/978-3-319-21155-8_4.

P34 Rose, Louis M. et al. (2014). "Epsilon Flock: a model migration language". In: *Software & Systems Modeling*. DOI: https://doi.org/10.1007/s10270-012-0296-2.

P39 Berramla, K., E. A. Deba, and M. Senouci (2015). "Formal validation of model transformation with Coq proof assistant". In: *2015 First International Conference on New Technologies of Information and Communication*. NTIC 2015. DOI: https://doi.org/10.1109/NTIC.2015.7368755.

P40 Legros, Elodie et al. (2009). "Generic and reflective graph transformations for checking and enforcement of modeling guidelines". In: *Journal of Visual Languages & Computing* 4. DOI: https://doi.org/10.1016/j.jvlc.2009.04.005.

P41 Sánchez Cuadrado, Jesús, Esther Guerra, and Juan de Lara (2011). "Generic Model Transformations: Write Once, Reuse Everywhere". In: *Theory and Practice of Model Transformations*. ICMT 2011. DOI: https://doi.org/10.1007/978-3-642-21732-6_5.

P43 Strüber, Daniel et al. (2017). "Henshin: A Usability-Focused Framework for EMF Model Transformation Development". In: *Graph Transformation*. ICGT 2017. DOI: https://doi.org/10.1007/978-3-319-61470-0_12.

P44 Wider, Arif (2014). "Implementing a Bidirectional Model Transformation Language as an Internal DSL in Scala". In: *EDBT/ICDT Workshops*. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.428.9439.

P45 Lawley, Michael and Kerry Raymond (2007). “Implementing a Practical Declarative Logic-based Model Transformation Engine”. In: *Proceedings of the 2007 ACM Symposium on Applied Computing*. SAC ’07. DOI: https://doi.org/10.1145/1244002.1244216.

P50 Liepiņš, Renārs (2012). “Library for Model Querying: IQuery”. In: *Proceedings of the 12th Workshop on OCL and Textual Modelling*. OCL ’12. DOI: https://doi.org/10.1145/2428516.2428522.

P52 Krikava, Filip, Philippe Collet, and Robert France (2014). “Manipulating Models Using Internal Domain-Specific Languages”. In: *Symposium On Applied Computing*. SAC ’14. DOI: https://doi.org/10.1145/2554850.2555127.

P56 Sun, Yu, Jules White, and Jeff Gray (2009). “Model Transformation by Demonstration”. In: *Model Driven Engineering Languages and Systems*. MODELS 2009. DOI: https://doi.org/10.1007/978-3-642-04425-0_58.

P58 Irazábal, Jerónimo and Claudia Pons (2010). “Model Transformation Languages Relying on Models as ADTs”. In: *Software Language Engineering*. SLE 2009. DOI: https://doi.org/10.1007/978-3-642-12107-4_10.

P59 Hebig, Regina et al. (2018). “Model Transformation Languages Under a Magnifying Glass: A Controlled Experiment with Xtend, ATL, and QVT”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018. DOI: https://doi.org/10.1145/3236024.3236046.

P60 Lara, Juan de et al. (2018). “Model Transformation Product Lines”. In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. MODELS ’18. DOI: https://doi.org/10.1145/3239372.3239377.

P63 Sendall, S. and W. Kozaczynski (2003). “Model transformation: the heart and soul of model-driven software development”. In: *IEEE Software*. DOI: https://doi.org/10.1109/MS.2003.1231150.

P64 Tratt, Laurence (2005). “Model transformations and tool integration”. In: *Software & Systems Modeling*. DOI: https://doi.org/10.1007/s10270-004-0070-1.

P66 — (2007). “Model transformations in MT”. In: *Science of Computer Programming*. DOI: https://doi.org/10.1016/j.scico.2007.05.003.

P70 Baar, Thomas and Jon Whittle (2007). “On the Usage of Concrete Syntax in Model Transformation Rules”. In: *Perspectives of Systems Informatics*. PSI 2006. DOI: https://doi.org/10.1007/978-3-540-70881-0_10.

P74 Sánchez Cuadrado, J., E. Guerra, and J. de Lara (2015). “Quick fixing ATL model transformations”. In: *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems*. MODELS ’15. DOI: https://doi.org/10.1109/MODELS.2015.7338245.

P75 Li, Dan, Xiaoshan Li, and Volker Stolz (2011). “QVT-based Model Transformation Using XSLT”. In: *SIGSOFT Softw. Eng. Notes*. DOI: https://doi.org/10.1145/1921532.1921563.

P77 Kusel, A. et al. (2015). “Reuse in model-to-model transformation languages: are we there yet?” In: *Software & Systems Modeling*. DOI: https://doi.org/10.1007/s10270-013-0343-7.

P78 Wimmer, Manuel et al. (2011). “Reusing Model Transformations across Heterogeneous Metamodels”. In: *ECEASST*. DOI: https://doi.org/10.14279/tuj.eceasst.50.722.

P80 Křikava, Filip, Philippe Collet, and Robert B. France (2014). “SIGMA: Scala Internal Domain-Specific Languages for Model Manipulations”. In: *Model-Driven Engineering Languages and Systems*. MODELS 2014. DOI: https://doi.org/10.1007/978-3-319-11653-2_35.

P81 Akehurst, D. H. et al. (2006). “SiTra: Simple Transformations in Java”. In: *Model Driven Engineering Languages and Systems*. MODELS 2006. DOI: https://doi.org/10.1007/11880240_25.

P86 Kolovos, Dimitrios S., Richard F. Paige, and Fiona A. C. Polack (2008). “The Epsilon Transformation Language”. In: *Theory and Practice of Model Transformations*. ICMT 2008. DOI: https://doi.org/10.1007/978-3-540-69927-9_4.

P90 Sánchez Cuadrado, Jesús, Esther Guerra, and Juan de Lara (2014). “Towards the Systematic Construction of Domain-Specific Transformation Languages”. In: *Modelling Foundations and Applications*. ECMFA 2014. DOI: https://doi.org/10.1007/978-3-319-09195-2_13.

P94 George, Lars, Arif Wider, and Markus Scheidgen (2012). “Type-Safe Model Transformation Languages as Internal DSLs in Scala”. In: *Theory and Practice of Model Transformations*. ICMT 2012. DOI: https://doi.org/10.1007/978-3-642-30476-7_11.

P95 Hinkel, Georg, Thomas Goldschmidt, et al. (2019). “Using internal domain-specific languages to inherit tool support and modularity for model transformations”. In: *Software & Systems Modeling*. DOI: https://doi.org/10.1007/s10270-017-0578-9.

P664 Agrawal, Aditya, Tihamer Levendovszky, et al. (2002). “Generative programming via graph transformations in the model-driven architecture”. In: *In OOPSLA 2002 Workshop in Generative Techniques in the context of Model Driven Architecture*. OOPSLA ’02.

URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.70.4824.

P665 Afimann, Uwe (1996). "How to uniformly specify program analysis and transformation with graph rewrite systems". In: *Compiler Construction*. CC 1996. DOI: https://doi.org/10.1007/3-540-61053-7_57.

P667 Radermacher, Ansgar (2000). "Support for Design Patterns through Graph Transformation Tools". In: *Applications of Graph Transformations with Industrial Relevance*. AGTIVE 1999. DOI: https://doi.org/10.1007/3-540-45104-8_9.

P669 Mohagheghi, Parastoo et al. (2013). "An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases". In: *Empirical Software Engineering*. DOI: https://doi.org/10.1007/s10664-012-9196-x.

P670 Staron, Miroslaw (2006). "Adopting Model Driven Software Development in Industry – A Case Study at Two Companies". In: *Model Driven Engineering Languages and Systems*. MODELS 2006. DOI: https://doi.org/10.1007/11880240_5.

P671 Panach, José Ignacio, Óscar Pastor, and Nathalie Aquino (2011). "A Model for Dealing with Usability in a Holistic MDD Method". In: *User Interface Description Language, Lisbon, Portugal*. UIDL '11.

P672 Amelunxen, Carsten et al. (2008). "Checking and Enforcement of Modeling Guidelines with Graph Transformations". In: *Applications of Graph Transformations with Industrial Relevance*. AGTIVE 2007. DOI: https://doi.org/10.1007/978-3-540-89020-1_22.

P673 Schmidt, Douglas (2006). "Guest Editor's Introduction: Model-Driven Engineering". In: *COMPUTER-IEEE COMPUTER SOCIETY*. DOI: https://doi.org/10.1109/MC.2006.58.

P674 Chafi, Hassan et al. (2010). "Language Virtualization for Heterogeneous Parallel Computing". In: *ACM Sigplan Notices*. DOI: https://doi.org/10.1145/1932682.1869527.

P675 Mernik, Marjan, Jan Heering, and Anthony M. Sloane (2005). "When and How to Develop Domain-specific Languages". In: *ACM computing surveys (CSUR)*. DOI: https://doi.org/10.1145/1118890.1118892.

P676 Gorp, Pieter Van and Louis M. Rose (2013). The Petri-Nets to Statecharts Transformation Case. DOI: https://doi.org/10.4204/EPTCS.135.3.

P677 Mens, Tom and Pieter Van Gorp (2006). "A Taxonomy of Model Transformation". In: *Electronic Notes in Theoretical Computer Science (GraMoT 2005)*. DOI: https://doi.org/10.1016/j.entcs.2005.10.021.

P800 Herndon, R. M. and V. A. Berzins (1988). "The realizable benefits of a language prototyping language". In: *IEEE Transactions on Software Engineering*. DOI: https://doi.org/10.1109/32.6159.

P801 Batory, Don, Jeff Thomas, and Marty Sirkin (1994). "Reengineering a Complex Application Using a Scalable Data Structure Compiler". In: *Proceedings of the 2Nd ACM SIGSOFT Symposium on Foundations of Software Engineering*. SIGSOFT '94. DOI: https://doi.org/10.1145/193173.195299.

P803 Kieburtz, Richard B. et al. (1996). "A Software Engineering Experiment in Software Component Generation". In: *Proceedings of the 18th International Conference on Software Engineering*. ICSE'96. DOI: https://doi.org/10.1109/ICSE.1996.493448.

P804 Gray, J. and G. Karsai (2003). "An examination of DSLs for concisely representing model traversals and transformations". In: *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*. HICSS '03. DOI: https://doi.org/10.1109/HICSS.2003.1174892.

References

1. Alves, R., Nunes, N.J.: Ceiling and threshold of paas tools: the role of learnability in tool adoption. In: International Conference on Human-Centred Software Engineering. HESSD 2016. (2016). https://doi.org/10.1007/978-3-319-44902-9_21
2. van Amstel, M.F., van den Brand, M.G.J.: Model transformation analysis: staying ahead of the maintenance nightmare. In: Theory and practice of model transformations. ICMT 2011. (2011). https://doi.org/10.1007/978-3-642-21732-6_8
3. Arendt, T. et al.: Henshin: advanced concepts and tools for in-place EMF model transformations. In: Model Driven Engineering Languages and Systems. MODELS 2010. (2010). https://doi.org/10.1007/978-3-642-16145-2_9
4. Aruoba, S.B., Fernandez-Villaverde, J.: A comparison of programming languages in economics. Technical report National Bureau of Economic Research, Inc. (2014). https://EconPapers.repec.org/RePEc:nbr:nberwo:20263
5. Auer, F., Felderer, M.: Current state of research on continuous experimentation: a systematic mapping study. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). (2018). https://doi.org/10.1109/SEAA.2018.00062
6. Badampudi, D., Wohlin, C., Petersen, K.: Experiences from using snowballing and database searches in systematic literature studies. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. EASE '15. (2015). https://doi.org/10.1145/2745802.2745818
7. Balogh, A., Varro, D.: Advanced model transformation language constructs in the VIATRA2 framework. In: Proceedings of the 2006 ACM Symposium on Applied Computing. SAC '06. (2006). https://doi.org/10.1145/1141277.1141575
8. Barat, S. et al.: A model-based approach to systematic review of research literature. In: Proceedings of the 10th Innovations in Software Engineering Conference. ISEC '17. (2017). https://doi.org/10.1145/3021460.3021462
9. Barb, A.S., et al.: A statistical study of the relevance of lines of code measures in software projects. In: Innovations in Systems and Software Engineering. (2014). https://doi.org/10.1007/s11334-014-0231-5

10. Basili, V., Reiter, R.: An investigation of human factors in software development. In: Computer. (1979). https://doi.org/10.1109/MC.1979.1658573
11. Basili, V.R., Caldiera, G., Dieter R.H.: The goal question metric approach. In: Encyclopedia of Software Engineering (1994)
12. Batory, D., Johnson, C., et al.: Achieving extensibility through product-lines and domain-specific languages: a case study. In: ACM Transactions on Software Engineering and Methodology (2002). https://doi.org/10.1145/505145.505147
13. Boehm, B., et al.: Cost models for future software life cycle processes: COCOMO 2.0. In: Annals of Software Engineering (1995). https://doi.org/10.1007/BF02249046
14. Boot, A., Sutton, A., Papaioannou, D.: Systematic Approaches to a Successful Literature Review. Sage, Thousand Oaks (2016)
15. Burkhardt, J.-M., Detiénne, F., Wiedenbeck, S.: Object-oriented program comprehension: effect of expertise, task and phase. In: Empirical Software Engineering (2002). https://doi.org/10.1023/A:1015297914742
16. Cabot, L., Burgueño, J., Gérard, S.: The future of model transformation languages: an open community discussion. In: Journal of Object Technology (2019). https://doi.org/10.5381/jot.2019.18.3.a7
17. Calvar, T., et al.: Efficient ATL incremental transformations. In: Journal of Object Technology (2019). https://doi.org/10.5381/jot.2019.18.3.a2
18. Cary, J.R., Shasharina, S.G., Cummings, J.C., Reynders, J.V., Hinker, P.J., et al.: Comparison of C++ and Fortran 90 for object-oriented scientific programming. Comput. Phys. Commun. **105**(1), 20–36 (1997)
19. Charmaz, K.: Constructing Grounded Theory. Sage, Thousand Oaks (2014)
20. Ciccozzi, F., Malavolta, I., Selic, B.: Execution of UML models: a systematic review of research and practice. In: Software & Systems Modeling (2019). https://doi.org/10.1007/s10270-018-0675-4
21. Cuadrado, J., Molina, J.G., Tortosa, M.M.: RubyTL: a practical, extensible transformation language. In: Model Driven Architecture–Foundations and Applications. ECMDA-FA 2006 (2006). https://doi.org/10.1007/11787044_13
22. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. In: IBM Systems Journal (2006). https://doi.org/10.1147/sj.453.0621
23. Dieste, O., et al.: Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study. In: Empirical Software Engineering (2017). https://doi.org/10.1007/s10664-016-9471-3
24. Fang, J., Varbanescu, A.L., Sips, H.: A comprehensive performance comparison of CUDA and OpenCL. In: 2011 International Conference on Parallel Processing. ICPP 2011 (2011). https://doi.org/10.1109/ICPP.2011.45
25. Frakes, W.B., Succi, G.: An industrial study of reuse, quality, and productivity. In: Journal of Systems and Software (2001). https://doi.org/10.1016/S0164-1212(00)00121-7
26. Galster, M., et al.: Variability in software systems—a systematic literature review. In: IEEE Transactions on Software Engineering (2014). https://doi.org/10.1109/TSE.2013.56
27. Gherardi, L., Brugali, D., Comotti D.: A Java vs. C++ performance evaluation: a 3D modeling benchmark. In: International Conference on Simulation, Modeling, and Programming for Autonomous Robots. SIMPAR 2012 (2012). https://doi.org/10.1007/978-3-642-34327-8_17
28. Hailpern, B., Tarr, P.: Model-driven development: the good, the bad, and the ugly. In: IBM Systems Journal (2006). https://doi.org/10.1147/sj.453.0451
29. Hebig, R., et al.: Model transformation languages under a magnifying glass: a controlled experiment with Xtend, ATL, and QVT. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2018 (2018). https://doi.org/10.1145/3236024.3236046
30. Henderson, R., Zorn, B.: A comparison of object-oriented programming in four modern languages. In: Software: Practice and Experience (1994). https://doi.org/10.1002/spe.4380241106
31. Hinkel, G.: An approach to maintainable model transformations with an internal DSL. PhD thesis. National Research Center (2013)
32. Hinkel, G., Burger, E.: Change propagation and bidirectionality in internal transformation DSLs. In: Software & Systems Modeling (2019). https://doi.org/10.1007/s10270-017-0617-6
33. Hinkel, G., Goldschmidt, T., et al.: Using internal domain-specific languages to inherit tool support and modularity for model transformations. In: Software & Systems Modeling (2019). https://doi.org/10.1007/s10270-017-0578-9
34. Hutchinson, John, et al.: Empirical assessment of MDE in industry. In: Proceedings of the 33rd International Conference on Software Engineering. ICSE '11 (2011). https://doi.org/10.1145/1985793.1985858
35. ISO/IEC 25010:2011 (2011). ISO/IEC. URL: https://www.iso.org/standard/22749.html
36. Jakumeit, E., et al.: A survey and comparison of transformation tools based on the transformation tool contest. In: Science of Computer Programming (2014). https://doi.org/10.1016/j.scico.2013.10.009
37. Jones, T.C.: Measuring programming quality and productivity. In: IBM Systems Journal (1978). https://doi.org/10.1147/sj.171.0039
38. Jouault, F., et al.: ATL: A QVT-like transformation language. In: Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications. OOPSLA '06 (2006). https://doi.org/10.1145/1176617.1176691
39. Kahani, N., et al.:. Survey and classification of model transformation tools. In: Software & Systems Modeling (2019). https://doi.org/10.1007/s10270-018-0665-6
40. Kieburtz, R.B., et al.: A software engineering experiment in software component generation. In: Proceedings of the 18th International Conference on Software Engineering. ICSE'96 (1996). https://doi.org/10.1109/ICSE.1996.493448
41. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering (2007). https://www.researchgate.net/publication/302924724_Guidelines_for_performing_Systematic_Literature_Reviews_in_Software_Engineering
42. Kofod-Petersen, A.: How to do a Structured Literature Review in computer science (2015). https://www.researchgate.net/publication/265158913_How_to_do_a_Structured_Literature_Review_in_computer_science
43. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The Epsilon transformation language. In: Theory and Practice of Model Transformations, ICMT 2008 (2008). https://doi.org/10.1007/978-3-540-69927-9_4
44. Kosar, T., Bohra, S., Mernik, M.: Domain-specific languages: a systematic mapping study. In: Information and Software Technology (2016). https://doi.org/10.1016/j.infsof.2015.11.001
45. Kosar, T., et al.: Comparing general-purpose and domain-specific languages: an empirical study. In: ComSIS–Computer Science an Information Systems Journal (2010). https://doi.org/10.2298/CSIS1002247K
46. Kfikava, F., Collet, P., France, R.B.: SIGMA: Scala internal domain-specific languages for model manipulations. In: Model-Driven Engineering Languages and Systems. MODELS 2014 (2014). https://doi.org/10.1007/978-3-319-11653-2_35
47. Kurniawan, B., Xue, J.: A comparative study of web application design models using the java technologies. In: Asia-Pacific Web Conference. APWeb 2004 (2004). https://doi.org/10.1007/978-3-540-24655-8_77

48. Loniewski, G., Insfran, E., Abrahão, S.: A systematic review of the use of requirements engineering techniques in model-driven development. In: Model Driven Engineering Languages and Systems (2010). https://doi.org/10.1007/978-3-642-16129-2_16
49. Mens, T., Van Gorp, P.: A taxonomy of model transformation. In: Electronic Notes in Theoretical Computer Science (GraMoT 2005) (2006). https://doi.org/10.1016/j.entcs.2005.10.021
50. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. In: ACM Computing Surveys (CSUR) (2005). https://doi.org/10.1145/1118890.1118892
51. Mohagheghi, P., et al.: An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. In: Empirical Software Engineering (2013). https://doi.org/10.1007/s10664-012-9196-x
52. OMG (2001). Model Driven Architecture (MDA), ormsc/2001-07-01
53. Prechelt, L.: An empirical comparison of seven programming languages. In: Computer (2000). https://doi.org/10.1109/2.876288
54. Rein, P., Taeumel, M., Hirschfeld, R.: Towards empirical evidence on the comprehensibility of natural language versus programming language. In: Design Thinking Research (2019). https://doi.org/10.1007/978-3-030-28960-7_7
55. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. In: IEEE Software (2003). https://doi.org/10.1109/MS.2003.1231150
56. Shaw, M.: Writing good software engineering research papers. In: 25th International Conference on Software Engineering, 2003. Proceedings (2003). https://doi.org/10.1109/ICSE.2003.1201262
57. Shevtsov, S., et al.: Control-theoretical software adaptation: a systematic literature review. In: IEEE Transactions on Software Engineering (2018). https://doi.org/10.1109/TSE.2017.2704579
58. Sjoberg, D.I.K., et al.: Conducting realistic experiments in software engineering. In: Proceedings International Symposium on Empirical Software Engineering. ISESE '02 (2002). https://doi.org/10.1109/ISESE.2002.1166921
59. Somasundaram, R., Karlsbjerg, J.: Research philosophies in the IOS adoption field. In: ECIS 2003 Proceedings, pp. 53 (2003)
60. Tratt, L.: Model transformations and tool integration. In: Software & Systems Modeling (2005). https://doi.org/10.1007/s10270-004-0070-1
61. Van Deursen, A., Klint, P.: Domain-specific language design requires feature descriptions. In: Journal of Computing and Information Technology (2002). https://doi.org/10.2498/cit.2002.01.01
62. Van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. In: ACM Sigplan Notices (2000). https://doi.org/10.1145/352029.352035
63. Varro, D., et al.: Termination analysis of model transformations by Petri Nets. In: Graph Transformations. ICGT 2006 (2006). https://doi.org/10.1007/11841883_19
64. Weyns, D., et al.: Claims and supporting evidence for self-adaptive systems: a literature study. In: Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS '12 (2012). https://doi.org/10.1109/SEAMS.2012.6224395
65. Whittle, J., et al.: Industrial adoption of model-driven engineering: are the tools really the problem?. In: Model-Driven Engineering Languages and Systems. MODELS 2013 (2013). https://doi.org/10.1007/978-3-642-41533-3_1
66. Wiger, U., Telecom Ab, E.: Four-fold Increase in Productivity and Quality -Industrial-Strength Functional Programming in Telecom-Class Products (2001)
67. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. EASE '14. Association for Computing Machinery (2014). https://doi.org/10.1145/2601248.2601268

Stefan Götz is a PhD student at the Ulm University. His research is focused on topics surrounding the development and evaluation of model transformation languages. Prior to his work as a PhD student, he was a student of software engineering at the Ulm University where he received his M.Sc. in.

Proof. Dr Mathias Tichy is full professor for software engineering at the Ulm University and director of Institute of Software Engineering and Programming Languages. His main research focus is on model-driven software engineering, particularly for cyber-physical systems. He works on requirements engineering, dependability, and validation and verification complemented by empirical research techniques. He is a regular member of programme committees for conferences and workshops in the area of software engineering and model-driven development. He is a co-author of over 110 peer-reviewed publications.

Raffaela Groner is a PhD student at the Ulm University. Her research is focused on the performance of model transformations. Prior, she studied computer science at the Ulm University.