Software & Systems Modeling
https://doi.org/10.1007/s10270-018-0665-6

REGULAR PAPER

Survey and classification of model transformation tools

Nafiseh Kahani[1] · Mojtaba Bagherzadeh[1] · James R. Cordy[1] · Juergen Dingel[1] · Daniel Varró[2,3]

Received: 5 May 2017 / Revised: 31 January 2018 / Accepted: 2 February 2018

Abstract
Model transformation lies at the very core of model-driven engineering, and a large number of model transformation languages and tools have been proposed over the last few years. These tools can be used to develop, transform, merge, exchange, compare, and verify models and metamodels. In this paper, we present a comprehensive catalog of existing metamodel-based transformation tools and compare them using a qualitative framework. We begin by organizing the 60 tools we identified into a general classification based on the transformation approach used. We then compare these tools using a number of particular facets, where each facet belongs to one of six different categories and may contain several attributes. The results of the study are discussed in detail and made publicly available in a companion website with a capability to search for tools using the specified facets as search criteria. Our study provides a thorough picture of the state-of-the-art in model transformation techniques and tools. Our results are potentially beneficial to many stakeholders in the modeling community, including practitioners, researchers, and transformation tool developers.

Keywords Model-driven development · Model transformation tools · Metamodel · Classification · Survey

1 Introduction

Model-driven engineering (MDE) is a growing field that advocates the use of models during the entire development process of a system. By leveraging abstraction and automation, MDE techniques can simplify communication and design activities, increase productivity and compatibility between systems, and boost development efficiency [1]. MDE can also facilitate a more comprehensive description of the system, since models can be used to describe different viewpoints.

A special case of MDE is model-driven development (MDD), a model-centric development approach in which models serve as primary artifacts from which, e.g., fully executable code is generated automatically. Model transformation lies at the heart of MDD, supporting model-to-text (M2T) transformations, which allow the transformation of models to textual artifacts such as code, reports, or documentation; and model-to-model (M2M) transformations, through which input models can be used to create different kinds of models in different languages and on different levels of abstraction. Model transformation can thus support a broad range of tasks including refinement, synthesis, abstraction, querying, translation, migration, analysis, refactoring, normalization, optimization, merging, debugging, and synchronization [2,3]. MDD-based systems have been widely used in the industry at large companies, such as Thales, Airbus, Boeing, and Ericsson.

Over the last two decades, an impressive amount of research effort has been invested in model transformation, and a large number of standards, languages, and tools have been proposed. As cataloged in this paper, more than 60 tools supporting model transformation have been presented

Communicated by Professor Alfonso Pierantonio.

✉ Nafiseh Kahani
kahani@cs.queensu.ca

Mojtaba Bagherzadeh
mojtaba@cs.queensu.ca

James R. Cordy
cordy@cs.queensu.ca

Juergen Dingel
dingel@cs.queensu.ca

Daniel Varró
daniel.varro@mcgill.ca

1 School of Computing, Queen's University, Kingston, Canada

2 School of Electrical and Computer Engineering, McGill University, Montreal, Canada

3 MTA-BME Lendület Research Group on Cyber-Physical Systems, Budapest, Hungary

Published online: 12 March 2018

in some form. These tools differ significantly in their capabilities, limitations, and requirements, making it difficult to select the tool that is most suitable for any given task. To facilitate the effective use of model transformation in practice and consolidate existing research and development results, a comparative survey of the state-of-the-art in model transformation *tools* would be useful. While a number of publications [3–15] have classified and compared model transformation *approaches*, none of them comprehensively addresses this need, either because they cover only a small number of tools, or because they compare them with respect to only a limited, predominantly technical, set of criteria.

To remedy this situation, this paper presents a comprehensive, up-to-date, comparative survey of the state-of-the-art in model transformation tools. It is based on an extensive study of 60 tools and classifies and compares them using 46 facets distributed over six categories: *general, model-level, transformation style, user experience, collaboration support,* and *runtime requirements*. The *general* category considers nontechnical facets, such as licensing and the availability of supporting resources. The *model-level* category focusses on aspects related to modeling, such as the support for different languages and standards (to express models and metamodels), repositories, and model management. The *transformation* category compares tools with respect to aspects of transformation implementation and execution such as rule scheduling, organization, and application control. The *user experience* category collects facets pertinent to the user-friendliness of the tool, such as editor features and support for typical development activities such as debugging, refactoring, and profiling. The *collaboration support* category groups facets related to collaborative development, interoperability, reuse, and extensibility. The *runtime requirements* category is related to assumptions the tool makes about its operating environment, such as operating systems, hosting frameworks, and IDEs.

In summary, our paper makes the following contributions:

1. *Classification and comparison* We present a catalog of all 60 model transformation tools surveyed using a broad classification of the general transformation approach underlying each tool (Sect. 4). We then conduct a fine-grained comparison of the tools with respect to 46 facets grouped into the six categories listed above (Sect. 5). To increase the accessibility and utility of the catalog and the comparison, we have made both publicly available on a website[1] with a search capability determining which tools support a given set of facets.
2. *Analysis of discontinued tools* We study discontinued tools in an attempt to determine factors that may have contributed to their discontinuation. We observe commonalities among these tools, such as the number of developers, the standards used, and the kinds of underlying transformation approaches supported. For example, more than half (6 of 11) of the tools based on a relational transformation language have been discontinued. More specifically, four out of nine tools targeting the QVT standard have been discontinued.
3. *Analysis of (un)supported facets* We summarize which facets are supported and which are unsupported overall. We study the relationship between the transformation approach and transformation language used, on the one hand, and the facets supported, on the other hand.

Our results are potentially useful to many stakeholders in the modeling community, including practitioners, researchers, and tool developers. Model transformation practitioners can use our catalog and comparison to determine the suitability of existing tools for planned transformation tasks, and our facets to evaluate and compare new tools. Our identification of facets that are poorly supported by existing tools might inspire modeling researchers to identify and tackle the underlying research challenges, using our catalog to help them identify appropriate tools to prototype potential solutions to these challenges. Model transformation tool developers can use our work to compare their tools with other existing tools, to improve their capabilities and support for certain facets.

The remainder of this paper is organized as follows. Section 2 presents a basic introduction to model-driven development. Section 3 provides an overview of the research method used in our study. Section 4 classifies tools based on transformation approaches, while Sect. 5 compares tools based on different facets. Section 6 highlights central observations and discusses results. Section 7 examines related work, and Sect. 8 concludes the paper.

2 Background

We begin with a basic introduction to model-driven terminology. Rothenberg et al. [16] define a model as follows: "A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality". Considering models as dynamic artifacts [17], it is possible to perform different operations on them, such as merging to integrate models and produce a new model, or refactoring to improve the internal structure of the model without changing its behavior or semantics.

Languages used to specify models can be graphical, textual, or both. There are two classifications for modeling languages [17]: (1) domain-specific modeling (DSM) lan-

[1] http://www.mdetools.com.

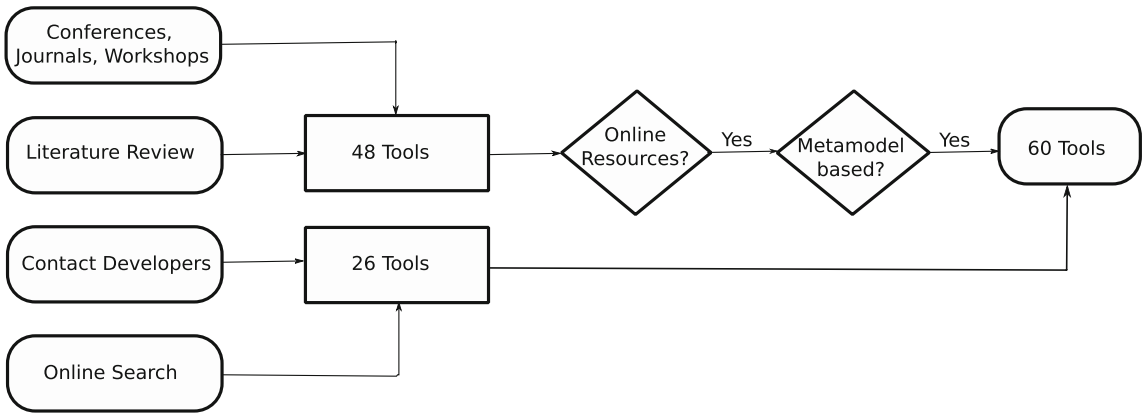

Fig. 1 Tool selection process

guages, which are dedicated to a particular domain or context for modeling purposes; and (2) general-purpose modeling (GPM) languages, such as the Unified Modeling Language (UML) [18], which can be applied to any domain. DSM languages have concepts relevant to the domain, and thus support higher-level abstractions than GPM languages, which typically makes them less complex and easier to use.

A modeling language is defined by its abstract syntax, its semantics, and its concrete syntax(es). The *abstract syntax* describes the structure and elements of the model, the properties and relations between the elements, and the validity constraints (i.e., well-formedness rules) of the model. Abstract syntax acts like a grammar for textual languages. In MDD, the abstract syntax is captured by a *metamodel*. We say that a model *conforms* to its metamodel, and that a metamodel *instantiates* models of its type. The *concrete syntax* can be textual, in the style of a programming language, graphical, using graphical icons to display the elements of the model and the relations between them, or both. Tools such as Graphiti [19] and Sirius [20] can specify graphical concrete syntax, and tools such as Xtext [21] and EMFText [22] can be used to describe textual concrete syntax. It is possible to have several different concrete syntaxes for one abstract syntax. In addition, concrete and abstract syntax are separate; thus, it is possible to apply the same concrete syntax for different abstract syntaxes. However, an abstract syntax does not specify the concrete notation or the meaning of relationships and other language concepts. Thus, a *semantics*, defined using denotational, operational, translational, or pragmatic approaches [23], is needed to describe the meaning of modeling elements and their combinations.

A *model transformation* is a program used to transform a model from one representation to another. The input of a transformation is called an *input model*, which conforms to a *source metamodel*, and its output is an *output model*, which conforms to a *target metamodel* (in M2M transformations), or *grammar* (in M2T transformations). A *model transformation definition* written in a model transformation language defines how one or more input model(s) are transformed to one or more output model(s). If the language of the transformation definition is rule-based, the transformation definition consists of a set of transformation rules. The *transformation engine* or *tool* uses the model transformation definition to produce output model(s) from input model(s).

3 Research method

The main contribution of this paper is a summary of the results of a study of 60 model transformation tools, organized into a new comprehensive catalog of tool-related facets. In the following, we discuss the approach that we followed to select the tools and facets, create the classification, and evaluate the tools using it.

3.1 Tool selection

As shown in Fig. 1, in order to select the tools for our study, we began by identifying the main conferences (i.e., the International Conference on Model Driven Engineering Languages and Systems (MODELS), the Transformation Tools Contest, the European Conference on Modelling Foundations and Applications (ECMFA), and the International Conference on Model Transformation (ICMT)), journals (i.e., Software and Systems Modeling (Springer) and Science of Computer Programming (Elsevier)), and workshops (i.e., the Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa), the Workshop on Domain-Specific Modeling (DSM), and the Workshop on Bidirectional Transformations (BX)) in the field, and reviewed previously published surveys of model transformation tools. 48 tools were identified in this first step. We then excluded tools that do not have a download page (e.g., ArcStyler [24], Yet Another Transformation Language/YATL [25], Codagen Architect [26], OptimalJ [27],

EMFTiger [28], FUUT-je [29], MOMENT [30], ATC [31], b+m ArchitectureWare [32], GenGen [33]), and tools that are not fully available or usable (e.g., QVTd [34] which is a partial implementation of QVTc and QVTr; and VMTL tool [35]). Our focus is on metamodel-based modeling tools, so we excluded tools such as WebRatio [36] and UMT [37] that are not metamodel-based. This second step resulted in a list of 34 tools.

In the third step, we used several search engines (hosted by *Google Scholar*, *Sourceforge*, and *Github*) and search terms ("model transformation tools", "model-to-model tools", "model-to-text tools", "M2M tools", "M2T tools" and "metamodel-based tools") to locate more tools. This step identified an additional 23 tools. We also contacted known tool developers via email and in person (e.g., at the MODELS conference) to ask for more tools. This step resulted in 3 more tools. Our final list includes 60 tools (Tables 1 and 2).

3.2 Facet selection

Inspired by the work of Roy et al. on the evaluation of clone-detection tools [38], we organized the features used to compare the model transformation tools into facets, each of which have different, but possibly overlapping attributes. Related facets are grouped into categories. Tables 3, 4, 5, 6, 7 and 8 list the facets pertinent to each category. The first column in each table shows the full name of the facet, the second column is related to the unique identifiers of the facet's attribute values, the third column provides a short description of the values, and the last column shows which percentage of tools support the facet's attributes.

To select the facets, we used the following three steps:

(1) We mined all 60 tool documents and their websites to determine their facets.We noticed that some of the assessed tools provide capabilities in addition to supporting the development of model transformations. For example, the Melange tool supports generic model transformations that can be applied to models conforming to different modeling languages, without having to change anything (based on model typing). Melange features language composition, slicing, and other mechanisms to ease development of DSM languages. The GROOVE tool is mainly used for editing graphs and graph transformation rules, and model checking graph transformations [39], so supports transformation as a by-product. In each case, we studied these additional facets to determine whether they could be expected to be supported by model transformation tools in general.

(2) We studied the related publications on classifying and comparing model transformation tools based on their features, such as [4,5,8–10,12], to make sure that the list of our facets is complete. The final results of the mentioned steps include 45 facets. To the best of our knowledge, 10 of these facets are novel and have not been used in the literature before. We label the previously facets with a reference to the corresponding publication. In many cases, the attribute values of shared facets are on a different level of granularity than in the original publication to allow for the information we collected in steps 1 and 2 of our facet selection process to be better reflected.

(3) We categorized the extracted facets into six categories (i.e., *general, model-level, transformation, user experience, collaboration support, and run-time requirements*).

3.3 Tool evaluation

The first author assessed the facets of the tools using all of the tools' available resources, including websites, tutorials, user manuals, forums, and published papers, as well as experience working with the tools, to extract the required information. The results were saved in an evaluation form and then re-checked by the second author. In cases where we were uncertain about a facet, we contacted the authors of the tools to clarify and to check that the other facets noted for their tools were correct. Tool developers replied and provided their feedback for 52 of the 60 tools. Table 9 shows the results of this overall evaluation.

4 Classification of tools

Based on the representations of the input and output models of the transformation, model transformation tools can be classified into three main categories: model-to-model (M2M), model-to-text (M2T) and text-to-model (T2M). The output of a M2M transformation is an instance of a target metamodel, whereas M2T approaches typically use target grammars to describe the structure of their textual output. T2M transformation tools, such as MoDisco [40], accept text representations as input and produce output models described by metamodels. T2M tools are most often used for reverse engineering, and are usually based on compiler technology (such as parser generation), rather than modeling technology (such as metamodeling). In this paper, we focus on metamodel-based approaches, that is, M2M (Sect. 4.1) and M2T (Sect. 4.2) transformations, and do not consider T2M further. Tools that support both M2M and M2T model transformations are discussed in both categories. Inspired by previous surveys [4,5] and to facilitate comparison, we categorize the tools based on their main functionality into two main groups, namely M2M and M2T. Figure 2 shows the corresponding classification tree.

Tables 1 and 2 provide a high-level overview of M2M and M2T tools, respectively, based on a classification of their transformation approach. The third column in each table provides a brief, informal description of the tool, and the fourth

Table 1 Classification of model-to-model (M2M) transformation tools

App.	Tool	Description	Lang.	FR	LR
Relational	UML-RSDS [41]	supports model-based development using UML	Java	2005	2017
	Tefkat [42]	a rule-, pattern-and template-based implementation of the Tefkat language	Java	2004	2008
	JTL[43]	a bidirectional model transformation tool supporting change propagation	ASP	2006	2015
	PTL [44]	ATL-style rules combined with logic rules to define transformations	Java	2013	2013
	ModTransf [45]	accepts models in XMI, XML or as a graph of objects	Java	2004	2005
	Echo [46]	supports model repair and transformation based on the Alloy model finder	Java	2013	2013
	QVTR-XSLT [47]	provides support for QVT relations in a graphical notation	Java	2009	2012
	ModelMorf [48]	fully supports the QVTr language	Java	2006	2006
	mediniQVT [49]	uses the QVTr language with a textual concrete syntax	Java	2007	2011
	PETE [50]	a Prolog rule-based tool supporting the transformation of Ecore models	Java	2009	2010
	TXL [51]	a grammar-based tool that can be used for model transformations	Turing+	1990	2017
Imperative	ModelAnt [52]	an extension of Apache ANT to support model transformations	Java	2004	2014
	Xtend[53]	statically-typed high-level programming tool for JVM, successor to Xpand	Java	2013	2017
	MetaEdit+ [54]	creates and develops DSM languages	MERL	1993	2017
	QVTo-Eclipse[55]	an Eclipse implementation of Borland Together based on QVTo	Java	2008	2017
	Kermeta2[56]	a meta-programming environment based on a model-oriented language	Java	2005	2012
	Modelio[57]	successor to Objecteering based on UML and BPMN	Java	2009	2017
	Umple[58]	a programming language family for model-oriented programming	Java	2008	2017
	Melange[59]	modular language workbench with re-usability, successor to Kermeta2	Java	2015	2017
	MagicDraw[60]	a visual UML, SysML, BPMN, and UPDM modeling tool	Java	1998	2017
	JAMDA[61]	supports Java code generation from a model of the business domain	Java	2002	2003
	SmartQVT[62]	a partial implementation of the QVTo language	Java	2006	2008
	SiTra[63]	a Java library supporting a Java-based approach to M2M	Java	2006	2012
	Mitra2[64]	successor to Mitra, optimized for semi-automated transformations	Java	2010	2012
	JQVT[65]	based on a compiled QVT engine for Java	Java	2012	2013
	Merlin[66]	based on EMF and Java Emitter Templates (JETs)	Java	2004	2005
	Together[67]	a set of Eclipse plugins to partially implement the QVTo language	Java	2003	2016
	MOFScript[68]	successor to UMT, implements the OMG MOFM2T specification	Java	2006	2011
Graph-based	GROOVE[69]	supports model checking graph transformation systems	Java	2003	2014
	UMLX[70]	supports concrete graphical syntax to complement the QVT language	Java	2005	2017
	AToM3[71]	a multi-paradigm modeling tool for visual languages	Python	2004	2008
	AToMPM [72]	successor to AToM3 that generates web-based DSM tools	Python	2012	2016
	AGG[73]	an algebraic approach attributed graph grammar transformation	Java	1997	2017
	BOTL[74]	a bi-directional transformation language with a precise formal foundation	Java	2003	2008
	GRoundTram[75]	a graph-based round-trip framework for bidirectional model transformations	OCaml	2009	2014
	eMoflon[76]	supports story-driven modeling and TGGs	Java	2006	2017
	MoTE[77]	provides bi-directionality, model synchronization and consistency	Java	2010	2016
	GReAT[78]	based on pattern specification, graph transf., and control-flow languages	VC++	2004	2014
	TGGInterpreter[79]	uses TGG rules to specify transformations	Java	2006	2011
	MOMoT[80]	a framework that combines modeling with search-based techniques	Java	2014	2016
	EMorF[81]	based on incremental TGG to support model synchronization	Java	2012	2012
	DSLTrans[82]	a visual language and tool for model transformations	Java	2011	2014
	MoTMoT[83]	graph rewriting based on a UML story diagrams	Java	2004	2006
Hybrid	VIATRA[84]	transformation framework focused on event-driven and reactive transf.	Java	2000	2017
	Eclectic[85]	based on family of model transformation languages	Java	2013	2013
	Epsilon[86]	a family of languages for a range of model management tasks	Java	2006	2016
	AGE[87]	based on the embedded DSLs with Ruby as host language	Ruby	2006	2010
	VMTS[88]	a domain-specific metamodeling and model processing framework	C#	2003	2014
	ATL[89]	model transformation language and execution environment based on EMF	Java	2005	2017
	Fujaba[90]	a story-driven modeling and graph transformation platform	Java	1994	2015
	GrGen.NET [91]	a programming productivity tool for graph transformations	Java	2003	2016
	Henshin[92]	successor to EMFTiger that supports visual modeling and transformation	Java	2011	2016
	Blu Age[93]	legacy application UML2 reverse engineering and code generation	Java	2006	2017
	MOLA [94]	a graphical procedural transformation language	Java	2005	2014
	SPARX[95]	a UML design and business analysis tool	C++	2000	2017
	MDWorkbench[96]	an Eclipse-based IDE for code generation and model transformation	Java	2005	2017

Table 2 Classification of the model-to-text (M2T) transformation tools

App.	Tool	Description	Lang.	FR	LR
Visitor	Kermeta2* [56]	a meta-programming environment based on a model-oriented language	Java	2005	2012
	Melange*[59]	successor to Kermeta2, modular language workbench with re-usability	Java	2015	2017
	JAMDA*[61]	supports Java code generation from a model of the business domain	Java	2002	2003
	AToM3*[71]	a multi-paradigm modeling tool for visual languages	Python	2004	2008
	AToMPM* [72]	successor to AToM3 that generates web-based DSM tools	Python	2012	2016
Template	ModelAnt* [52]	an extension of Apache ANT to support model transformations	Java	2004	2014
	ModTransf*[45]	accepts models in XMI, XML or as a graph of objects	Java	2004	2005
	Umple*[58]	a programming language family for model-oriented programming	Java	2008	2017
	Acceleo[97]	a pragmatic implementation of the OMG MOFM2T standard	Java	2006	2017
	MagicDraw*[60]	a visual UML, SysML, BPMN, and UPDM modeling tool	Java	1998	2017
	AGE*[87]	based on the embedded DSLs with Ruby as host language	Ruby	2006	2010
	eMoflon*[76]	supports story-driven modeling and TGGs	Java	2006	2017
	Henshin*[92]	successor to EMFTiger that supports visual modeling and transformation	Java	2011	2016
	AndroMDA[98]	a framework that transforms UML models into deployable components	Java	2003	2012
	Fujaba*[90]	a story-driven modeling and graph transformation platform	Java	1994	2015
	TXL*[51]	a grammar-based tool that can be used for model transformations	Turing+	1990	2017
	SPARX*[95]	a UML design and business analysis tool	C++	2000	2017
	Merlin*[66]	based on EMF and Java Emitter Templates (JETs)	Java	2004	2005
	MOFScript*[68]	successor to UMT, implements the OMG MOFM2T specification	Java	2006	2011
	Together*[67]	a set of Eclipse plugins to partially implement the QVTo language	Java	2003	2016
	Xpand[99]	a domain-specific M2T transformation framework for EMF models	Java	2004	2015
	Epsilon*[86]	a family of languages for a range of model management tasks	Java	2006	2016
	VIATRA*[100]	transformation framework focused on event-driven and reactive transf.	Java	2000	2017
	MDWorkbench*[96]	an Eclipse-based IDE for code generation and model transformation	Java	2005	2017
Hybrid	Actifsource[101]	a domain-specific tool to genereate code from software specification	Java	2010	2017
	MetaEdit+*[54]	creates and develops DSM languages	MERL	1993	2017
	Blu Age*[93]	legacy application UML2 reverse engineering and code generation	Java	2006	2017
	VMTS*[88]	a domain-specific metamodeling and model processing framework	C#	2003	2014
	Xtend*[53]	statically-typed high-level programming tool for JVM, successor to Xpand	Java	2013	2017
	GrGen.NET* [91]	a programming productivity tool for graph transformations	Java	2003	2016
	Modelio*[57]	successor to Objecteering based on UML and BPMN	Java	2009	2017

* (also M2M tool)

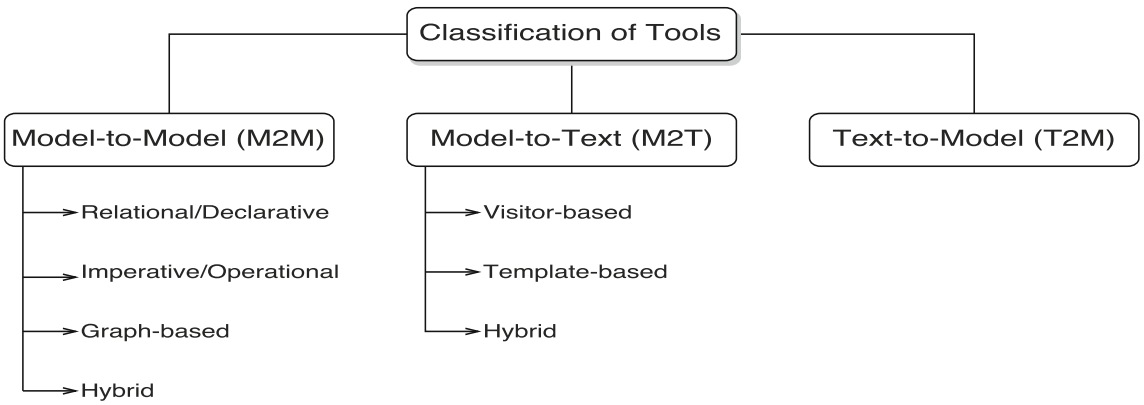

Fig. 2 High-level classification of tools

column shows the language in which it is implemented. The columns labeled *FR* and *LR* show the dates of first and latest release of tools up to the date of this study (2017). We classify the M2M and M2T tools according to their transformation approaches, as described in the following sections.

4.1 Model-to-model (M2M) tools

M2M model transformation tools convert one or more input models (conforming to a source metamodel) into one or more output models (conforming to a target metamodel). We classify M2M tools based on their transformation approach, which describes the set of language

constructs or mechanisms used to describe and apply transformations. M2M model transformation approaches can be divided into relational/declarative (Sect. 4.1.1), imperative/operational (Sect. 4.1.2), graph-based (Sect. 4.1.3), and hybrid (Sect. 4.1.4).

4.1.1 Relational/declarative approaches

Transformation specifications in languages and tools using this paradigm focus on *which* input elements should be transformed and their corresponding output elements (e.g., by specifying predicates or input-output constraints), without directly specifying *how* this transformation should be executed. In other words, relational approaches are based on defining relationships between the elements in the input and output models. These relationships are typically described using mathematical relations represented as predicates or constraints. Languages implementing the relational/declarative transformation paradigm can be either functional or logical. A logical transformation language has features that are well-suited to relational approaches such as searching, constraint propagation, and backtracking. In functional languages, the transformation from input(s) to output(s) is described as the composition of a set of functions mapping input model elements to corresponding outputs. UML-RSDS, Tefkat, JTL, PTL, ModTransf, PETE, and TXL are examples of tools that use relational approaches.

QVT Relational (QVTr) [102] is an example of a relational model transformation language that is supported by several tools (e.g., Echo, QVTR-XSLT, ModelMorf, mediniQVT). In QVTR, a relation is specified using two or more *domains*, with a pair of *when* and *where* clauses. Each domain represents a (partial) model in the transformation. The when clause determines the conditions under which the transformation relationship holds, and the where clause determines the conditions that must be met by all model elements for it to apply. Patterns are used to define the domains, which can be marked as either *check-only* or *enforced*. In check-only mode, the consistency of the output model elements is checked. In the case of a false result, the rule is enforced by modifying the elements of output model to make the output consistent with the input model. QVTr has both a textual and graphical concrete syntax. Echo and mediniQVT are tools based on the QVTr syntax, but use a semantics that departs from the Object Management Group (OMG) standard.

QVT Core (QVTc) [102] is a simple, low-level relational language based on pattern matching over a set of variables. The language evaluates conditions over these variables according to a set of models. QVTr is defined on top of QVTc; thus, a transformation in the core language is defined as a set of mappings from QVTr to QVTc. QVTr is defined at a higher-level of abstraction and supports more complex pattern matching than QVTc.

4.1.2 Imperative/operational approaches

Imperative languages focus on *how* and *when* the transformation should be executed, without drawing attention to the relations that must hold between source and target elements. Imperative languages specify transformations as a sequentially executed list of actions or rules. The language concepts and constructs of imperative transformation languages are similar to those in general-purpose imperative programming languages [12], so they typically are easy for the developers to learn. Procedural model transformation languages used in MetaEdit+ are imperative and use procedures to wrap frequently used sequences of actions.

QVT Operational (QVTo) [102] is an example of an imperative transformation language comparable to conventional procedural languages such as *C*. QVTo is supported by several tools, including QVTo-Eclipse, MagicDraw, SmartQVT, and Together. In QVTo, transformations are specified as a set of mappings, where each mapping transforms one (or more) input model element(s) to one (or more) output model element(s). QVTo mappings, similarly to QVTr relations, can have *when* and *where* clauses to limit their application.

The *direct manipulation* approach [4] is similar to the imperative approach, but with lower-level constructs and language concepts to support model transformations. In this approach, a general-purpose programming language such as Java or Visual Basic is augmented with advanced capabilities offered by the Application Programming Interface (API) of libraries allowing them to implement model transformations. The APIs allow users to create, manipulate and access the internal structure of models and metamodel instances directly. This approach is simple and developers do not need to learn a new language to write transformations. However, these languages were not primarily designed for direct model manipulation, so users must manually implement many of the required features of model transformations, such as traceability. Furthermore, dependence on particular APIs can impose restrictions on the transformations that can be supported. Examples of the tools that use the direct manipulation approach are JAMDA and SiTra. Examples of imperative tools are ModelAnt, Xtend, Kermeta2, Modelio, Umple, Melange, Mitra2, JQVT, Merlin, and MOFScript.

4.1.3 Graph-based approaches

Transformation languages following the graph-based paradigm are based on algebraic graph transformation [103], and represent the input and output models using variations of typed, attributed graphs. A graph transformation consists

of a set of graph transformation rules (also called rewriting rules or production rules), which are applied to an input (host) graph to produce an output graph. Each rule consists of a *Left-Hand Side* (LHS) graph, a *Right-Hand Side* (RHS) graph, and an optional *Negative Application Condition* (NAC) graph. The LHS specifies the model subgraph to which the rule can be applied, while the RHS specifies the corresponding new model subgraph. NACs are used to specify forbidden patterns in the host graph, for instance the absence of particular vertices and edges, in order to limit the application of the rule to the intended contexts. Most graph transformation approaches, such as GROOVE and AGG, allow specification of NACs for rules. When a graph transformation rule executes on a given host graph, subgraph matching is performed to look for matches of the rule's LHS in the host graph that also do *not* match the NAC, if any. Trying to match a subgraph is called pattern matching or evaluating a rule. If a match is found, the following steps are carried out: (a) host graph elements that appear in the rule's LHS but not in its RHS are deleted, (b) elements that appear only in the rule's RHS but not in the LHS are created and embedded into the host graph, and (c) host graph elements that occur in both the rule's LHS and RHS are preserved. The LHS and NACs specify the preconditions for the graph transformation rule's execution, while the RHS specifies the post-condition. Examples of tools in this category are AToMPM, GROOVE, UMLX, AToM3, AGG, BOTL, GRoundTram, GReAT, MOMoT, DSLTrans, and MoTMoT.

Graph-based languages are well suited to perform in-place and endogenous transformations, but have difficulty retaining traceability between input and output elements, requiring traceability links to be explicitly encoded. Triple Graph Grammars (TGGs) [104] were proposed to overcome this disadvantage by using *correspondence* graphs or metamodels that maintain an N-to-N relationship including tracing information between input and output elements. Thus, they can be used to synchronize two different models and check whether they are consistent. TGGs consist of three graphs, a source graph (left-hand), a target graph (right-hand), and a correspondence graph. TGGs are similar to QVTr but have a stronger theoretical foundation. In QVTr the dependencies of transformation rules are explicitly formulated in the *when* and *where* clauses, while the order of TGG rules is implicitly specified based on the satisfaction of their preconditions. Examples of TGG-based tools are eMoflon, MoTE, TGGInterpreter, and EMorF.

4.1.4 Hybrid approaches

Each technique has its own strengths and weaknesses. In imperative approaches, the programmer has detailed control over execution of the transformation, which allows for an efficient implementation of complex transformations. However, this explicit control can require more code, which can make the transformation more difficult to author, read and understand. Relational approaches, by contrast, can be more concise and easier to understand due to the expression of the transformation at a higher level of abstraction, with fewer implementation details. However, by relieving the developer from dealing with explicit control flow, relational transformation languages can be less expressive, and thus less suitable for complex transformation tasks. Expressing a model transformation in terms of visual graph transformation rules can be a challenging task [13].

Hybrid approaches attempt to circumvent these issues by allowing developers to mix-and-match constructs from different approaches when developing model transformations. For instance, MOLA, Fujaba, and GrGen.NET use a combination of graph-based and imperative approaches. ATL uses a combination of relational and imperative approaches. VIATRA and VMTS use relational, imperative and graph-based approaches. Epsilon and Eclectic provide a family of model transformation languages in which many types of approaches are intended to be supported. Other tools that support hybrid approaches for building model transformations include AGE (RubyTL), MDWorkbench, SPARX (Enterprise Architect), Henshin, and Blu Age.

4.2 Model-to-text (M2T) tools

M2T transformation tools transform one (or more) input model(s) into text (e.g., source code, documentation, or configuration files). We distinguish M2T tools based on their underlying implementation approach, which can be visitor-based (Sect. 4.2.1), template-based (Sect. 4.2.2), or hybrid (Sect. 4.2.3) [4].

4.2.1 Visitor-based approaches

Visitor-based approaches are similar to direct manipulation approaches, in the sense that they traverse a tree-based internal representation of the input model to generate code or other information from visited model elements. The generated text is written to a text output stream as elements are visited. The order of the traversal and thus the output text to generate is defined by the transformation rules. Some parts of the transformation, such as instructions for outputting the text, must be written manually by the user. Examples of tools in this category are Kermeta2, Melange, JAMDA, AToM3, and AToMPM.

4.2.2 Template-based approaches

In template-based M2T approaches, a template is used to specify the text to be generated by the transformation for the

given input models. A template is composed of static text (i.e., target text that will be generated in common for any given input model) and placeholders for data to be extracted from the input model. There is a meta-program for the dynamic part, which accesses the stored information in the models. An example of this approach is the MOF Model to Text Transformation Language (MOFM2T), which facilitates template composition and modular organization to handle complex M2T transformations.

Compared to visitor-based approaches, template-based M2T transformations tend to be easier to understand due to the similarity between templates and the generated code. Template-based approaches are currently much more popular than visitor-based, possibly because of the influence of modeling standards such as MOFM2T. Examples of template-based tools are ModelAnt, ModTransf, Umple, Acceleo, MagicDraw, AGE (RubyTL), eMoflon, Henshin, MDWorkbench (TGL), AndroMDA, Fujaba, TXL, SPARX (Enterprise Architect), Merlin, MOFScript, Xpand, Together, VIATRA, and Epsilon (EGL). The aspect-oriented programming constructs provided by Xpand can be used to enhance the modularity and reusability of templates.

4.2.3 Hybrid approaches

In cases where the generated output of the code generation heavily depends on the structure or information from the model [105], visitor-based approaches are typically a better choice than template-based ones. However, the visitor-based approach is not suitable when a considerable portion of the text to be generated is static and independent of the model content. Therefore, template-based approaches can be combined with the visitor pattern to design and implement M2T tools. Examples of hybrid M2T transformation tools include Actifsource, MetaEdit+ (MERL), Blu Age, VMTS, Xtend, GrGen.NET, and Modelio. In programming language-based tools, such as GrGen.NET and TXL, users can also write their own visitor-based transformations and templates.

5 Comparison of tools

In this section, we assess the 60 tools with respect to 46 facets organized into six categorizes, namely *General* (Sect. 5.1), *Model-level* (Sect. 5.2), *Transformation* (Sect. 5.3), *User Experience* (Sect. 5.4), *Collaboration Support* (Sect. 5.5), and *Runtime Requirements* (Sect. 5.6). Tables 3, 4, 5, 6, 7 and 8 show the facets in each category along with their attributes and the proportion of surveyed tools supporting each attribute. Table ("Appendix A") 9 summarizes our assessment of the tools.

5.1 General category

The *General* category considers nontechnical facets, such as licensing and the availability of support, which can be crucial factors when selecting a tool, particularly for use in commercial applications (Table 3).

Update Time (UP) This facet is motivated by importance of access to the latest changes and stable releases of the tool. We consider the update time to be *regular* if the tool is updated at least once a year (e.g., UML-RSDS, TXL, Xtend, Together, Actifsource, SPARX). If the tool does not follow a regular update schedule, we indicate its update time as *sometimes* (e.g., ModelAnt, GROOVE, VIATRA, Epsilon, MOLA, Fujaba). We consider the update time to be *discontinued* if the tool is no longer offered or is no longer being updated (e.g., JAMDA, Merlin, JQVT, Tefkat, ModTransf, AGE, ModelMorf, QVTR-XSLT, BOTL). The update time of these tools is usually limited to the initial development period, with only rare updates after the first release. Similarly, we describe the update time as *replaced*, if the tool was replaced by a successor. For example, AToMPM is the web-based successor of AToM3, that runs in the cloud to provide web-based modeling tools with a graphical user interface. Xpand has been discontinued; however, all its useful facets have been implemented in Xtend, which provides debugging, better performance, and tool support. Kermeta2 was eventually replaced with a newer version, Kermeta3, and merged with Melange, in which model transformation is supported as a core part of the DSM creation activities.

Licence (L) [8] This facet is motivated by the importance of *licensing* for distribution, redistribution, and modification of the tools, specifically for use in commercial applications. We found four different types of license, including open source, free without source in binary form, commercial, and commercial with free evaluation. The *open-source* tools can be provided under one of several well-known licenses, including the Eclipse Public License (EPL), the General Public License (GPL), the Apache License (ASL), the Berkeley Software Distribution (BSD) license, or the Massachusetts Institute of Technology (MIT) license. The main difference between these licenses is in the redistribution terms, which can affect the use of the tools in commercial applications. For example, code licensed under the GPL requires that any modified code be open-source, which limits use in industry if the code is to be proprietary. A detailed review examining the effect of open-source licenses on commercial software development can be found in Pearson [106]. Examples of tools using the EPL license are Echo, ModelAnt, Xtend, QVTo-Eclipse, Kermeta2, Melange, SmartQVT, JQVT, MOFScript, AGG, Henshin, VIATRA, Epsilon, ATL, Acceleo, Merlin, and Xpand. Examples of tools using the GPL license are Tefkat, AToMPM, MoTMoT, BOTL, Eclectic, and GrGen.NET. JAMDA and Groove are examples of tools

Table 3 Facets in the general category

Facet	Abbr.	Attributes		Percentage
Update Time (UP)	a	The tool is updated regularly		33%
	b	The tool is updated sometimes		35%
	c	The tool has been discontinued		27%
	d	The tool has been replaced by a successor		5%
Licence (L)	a	Open-source	Eclipse Public License (EPL)	42%
	b		General Public License (GPL)	13%
	c		Apache License (ASL)	3%
	d		Berkeley Software Distribution (BSD)	2%
	e		Massachusetts Institute of Technology (MIT)	5%
	f		Other licences	12%
	g	The tool is freely available in binary form		20%
	h	The tool is commercially available		17%
	i	The tool has a free evaluation licence		10%
Targeted Audiences (TA)	a	The tool has been used in academy		92%
	b	The tool has been used in industry		37%
Technical Support (TES)	a	The tool provides professional support		15%
	b	The tool provides limited support		38%
Supporting Resources (SR)	a	The tool provides documentation		88%
	b	The tool provides examples		93%
	c	The tool has a wiki-page		43%
	d	The tool has a website		92%
	e	The tool has a forum/community		38%
Security (SEC)	a	Obfuscation		14%
	b	Read-only/Locked models		19%

under the Apache license, and Umple, Fujaba, and GRoundTram are examples of tools with the MIT license. Some tools are dual-licensed such as eMoflon (GPL and EPL), and AndroMDA uses a BSD license. Examples of tools with other licenses are: *free without source in binary form* (e.g., UML-RSDS, PETE, TXL, MOLA, SPARX, VMTS), *commercial* (e.g., Blu Age, Together, MagicDraw, SPARX), and *commercial with free evaluation license* (e.g., Together, SPARX, MDWorkbench, MagicDraw)

Targeted Audiences (TA) [13,107] This facet refers to where the tools have been or are intended to be used: in academia, in industry, or both. *Academic* tools are typically developed to demonstrate theoretical concepts or to address new research challenges of interest to other researchers. On the other hand, *industrial* tools are designed to address existing industrial requirements, which can make them more practical for use in both industry and academia. A large number of tools, such as GROOVE, UMLX, AToMPM, BOTL, MOMoT, ModTransf, are primarily used in academia. Tools such as ATL, Epsilon, MagicDraw, Together, GrGen.NET, MDWorkbench, VIATRA, SPARX, Actifsource, Acceleo, MetaEdit+ are used in both academia and industry.

Technical Support (TES) This facet refers to the available resources for maintenance and technical support of the tool, including forums (offered by, e.g., TXL, Epsilon), mailing lists (available in, e.g., Tefkat, Together, Actifsource), bug and fix tracking (supported by, e.g., GrGen.NET, GROOVE), and so on. Users are more confident if they know that their problems with a tool will be considered and addressed. Open-source tools, which form the majority of model transformation tools, rely primarily on community support. However, developer support for adding new features, fixing bugs, and related software maintenance issues remains important for tool users. We use *limited-support* (supported in, e.g., Tefkat, ModelAnt, Xtend, GROOVE, UMLX) to describe limited online support, such as asking questions and reporting bugs, without an ongoing obligation on the part of the tool developers to fix bugs or answer questions.

Supporting Resources (SR) [8] This facet captures the availability of resources for learning and using the tool. Recent work on adoption issues in Eclipse modeling tools [108] shows that a lack of supporting resources, such as documentation, is the most common problem for both general and novice users of modeling tools. Examples of good supporting resources listed in Table 3 are tutorials and user guides (available for, e.g., UML-RSDS, GrGen.NET, eMoflon, TXL, Actifsource), completely worked out examples (available for, e.g., ATL, MDWorkbench, GROOVE), user forums (provided by, e.g., VIATRA, Umple, MetaEdit+), wiki pages (provided by, e.g., Henshin, eMoflon), and websites describing the status of the tool (available for, e.g., AToMPM, MagicDraw).

Security (SEC) This facet refers to the ability to limit user access to models or code, or to prevent accidental changes or updates to the referenced models. In addition to obfuscation (i.e., hiding sensitive information in a confidential model or generated code), and read-only/locked models, there are other types of security, such as role-based access control (sup-

ported in, e.g., Modelio, MetaEdit+) that use user rights to control access to different parts of a transformation, such as the metamodel. It is also important that access to repositories and the running of a code generator can be secured properly. In MetaEdit+, the generated code is obfuscated, and variables and function names are generally produced from the model text with a user-defined translator. Epsilon supports read-only Eclipse Modeling Framework (EMF) models to prevent accidental modification. In ATL, transformation input models can be navigated, but changes are not allowed.

5.2 Model-level category

The *Model-level* category deals with modeling aspects of the tools, such as support for meta-modeling and model repositories (Table 4). These facets are not related to the transformation authoring process; rather they speak to the support for related artifacts and activities.

Modeling Languages (ML) [17] Some tools provide an environment for modeling in different modeling languages, either a domain-specific DSM or a more general GPM. This facet considers which languages are supported by the assessed tools. Among the GPM languages, UML is by far the most popular. Different tools provide support for different *UML* diagrams. For example, UML-RSDS supports UML2.x for class, use case, state machine, activity, and sequence diagrams; MagicDraw provides full support for UML2.x and UML1.x; SPARX supports UML2.x for use case, activity, state machine, interaction overview, sequence diagrams, communication, package, class, object, composite, component and deployment diagrams; Blu Age supports UML2.x for class, activity, use case diagrams; AToMPM, Umple, and Actifsource support UML2.x for class and state diagrams; VMTS originally supported UML2.x for class, activity, use case, sequence, component, and deployment diagrams; Fujaba supports UML2.x for class, activity, and object diagrams; and Together supports UML2.x and UML1.x for class, activity, component, composite structure, deployment, state machine, use case, sequence, and communication diagrams. Some tools, such as Modelio, MagicDraw, MetaEdit+, and SPARX support the *Systems Modeling Language (SysML)* [109], a profile of UML. A UML profile is an extension of UML with additional semantics that specializes UML to a particular domain through constraint and extension mechanisms. Other examples of UML profiles include *executable UML (xtUML)* [110] and *UML for Real-Time* (UML-RT) [111].

Also supported are *Petri Nets* [112], a graphical formal modeling language, supported by tools such as AToM3, and MetaEdit+. Petri Nets consist of places, transitions, and arcs, where places are connected to transitions by input and output arcs. *Programming languages*, particularly object-oriented (OO) languages, can be used in some tools (e.g., VMTS, GrGen.NET, Blu Age) to describe models in a textual notation. The *Business Process Model and Notation (BPMN)* [113] (supported in, e.g., Modelio, MagicDraw, Together, SPARX) is a standardized graphical language for specifying business processes.

Some of the surveyed tools support other modeling languages. For example, GrGen.NET has its own model description language, and AGG supports attributed typed graphs. The modeling languages supported in Modelio are TOGAF, UPDM, SOAML, and UTP, whereas SPARX supports BPEL, UPDM, TOGAF, SOAML, and SOMF.

Metamodeling Languages (MML) [12,17] Input and output models of a transformation must conform to metamodels. This facet considers which metamodel technology can be used in the tools. The Meta-Object Facility (MOF) [114] is an OMG standard for defining metamodels, divided into essential MOF (eMOF) and complete MOF (cMOF). eMOF (supported in, e.g., Tefkat, Fujaba, UML-RSDS, ModTransf) is a simple core framework based on a subset of UML class diagrams. cMOF provides more sophisticated features and a graphical notation to specify more complex modeling languages, such as UML itself. Ecore [115] (supported in, e.g., PTL, JQVT, Echo, GROOVE), proposed by EMF, is another metamodeling notation based on the eMOF specification. Kernel MetaModel (KM3) [116] (supported in, e.g., GRoundTram, ATL) is a subset of Ecore to write metamodels using a textual representation. The use of MOF or Ecore can facilitate tool interoperability. In addition to these standards, many other metamodeling languages are used by various tools. Examples include: GOPPRR (MetaEdit+), ArkM3 (AToMPM), Genmodel (Henshin), MOLA MOF (MOLA), VPM (VIATRA before June 2015), VMTS Root (VMTS), MetaModeler (ModelMorf), and Umple (Umple).

Model Comparison (MC) [14,17] This facet refers to support for identifying the similarities and differences between homogeneous and heterogeneous models used by model transformation developers for tasks such as testing and model evolution management. A detailed review and classification of model comparison methods based on the types of models they can compare can be found in Stephan et al. [117]. Tools that support model comparison include ModelAnt, MetaEdit+, Umple, SPARX, Together, Epsilon (ECL), and Blu Age. Such tools can show the comparison results in terms of added/removed model elements/attributes (supported by, e.g., ModelAnt) or visually in tree form (supported by, e.g., SPARX). GRoundTram supports simple node/edge level comparison and binary bi-similarity comparison. Epsilon (ECL) follows a metamodel-independent approach using similarity-based matching strategies. MOMoT and VIATRA can support Ecore model comparison through the EMF-compare Eclipse plugin, which provides comparison and merge facilities for EMF models.

Table 4 Facets in the model-level category

Facet	Abbr.	Attributes		Percentage
Modeling Languages (ML)	a	GPM languages	UML 1.x	5%
	b		UML 2.x	18%
	c		xtUML	2%
	d		SysML	7%
	e	DSM languages	Petri Nets	3%
	f		Programming languages	7%
	g		BPMN	8%
	h		Other modeling languages	7%
MetaModeling Languages (MML)	a	The tool supports EMOF		23%
	b	The tool supports CMOF		
	c	The tool supports Ecore/EMF		63%
	d	The tool supports KM3		3%
	e	The tool supports other metamodeling languages		22%
Model Comparison (MC)	a	The tool compares homogeneous models		12%
	b	The tool compares heterogeneous models		3%
	c	Results are in visual/model forms (e.g., UML)		10%
	d	Results are in textual forms		5%
Model Query (MQ)	a	The tool supports model query		28%
MetaModeling Env. (MME)	a	The tool is a metamodeling tool		12%
Model Repositories (MR)	a	The tool supports EMF		58%
	b	The tool supports MDR		10%
	c	The tool supports other model repositories		18%
Compatibility with Stan. (CS)	a	The tool supports XMI		85%
	b	The tool supports CWM		3%
	c	The tool is an implementation of QVTo		8%
	d	The tool is an implementation of QVTr		8%
	e	The tool is an implementation of QVTc		3%
	f	The tool is an implementation of QVT-Like		15%
	g	The tool supports MOFM2T		7%
	h	The tool supports OCL expression		58%
	i	The tool supports DD specification		8%
	j	The tool supports HUTN		5%
	k	The tool supports JMI		8%
	l	The tool supports CMI		
Reverse Engineering (RE)	a	A reverse engineering tool		18%
Round-trip Engineering (RT)	a	A round-trip engineering tool		15%

Model Query (MQ) [3] Model query is an important capability in model transformation tools, used to request specific contents or a selection of the model elements. Model query can be categorized according to its supporting query language as: Graph patterns or OCL based. OCL is a standard declarative model query language mainly used in industry [118]. While query languages based on graph patterns are similar to logic programming, where the order of model exploration is freely determined by the query engine at evaluation time [118]. Querying can be also used for other purposes, such as model analysis and evolution, and reporting. Some tools provide model query as a standalone feature retrieving information without any side-effects on the models. Examples of tools that support model query are MagicDraw, EMorF, VMTS, MetaEdit+, and QVTR-XSLT. In EMorF, queries for model elements, such as objects and links (i.e., model patterns), can be expressed graphically. In MetaEdit+, queries can be made using the MERL generator, which is also used to access the models.

MetaModeling Environment (MME) [13] This facet indicates whether the tool can be used to define metamodels. Tools that include this facility are Melange, MetaEdit+, Kermeta2, AToM3, AToMPM, and VMTS. VMTS has a general-purpose metamodeling environment that supports an arbitrary number of metamodel levels. In Kermeta2, metamodels can be expressed in Kermeta2 itself. An overview of metamodeling in MDD can be found in [119].

Model Repositories (MR) [14] Models need to be stored and loaded to/from storage as files or repositories. Repositories such as the Eclipse Modeling Framework (EMF) and the NetBeans Meta-data Repository (MDR) are used to store large models, using an API to access and manipulate the models. *EMF*, a Java modeling framework and code generation facility based on a structured data model, is widely used by transformation tools (e.g., in JTL, PTL, PETE, MDWork-

bench, TGGInterpreter, Actifsource, Mitra2, MoTE, Merlin, Melange).

Other model repositories built on the EMF framework include Connected Data Objects (CDO) [120], ModelBus [121], EMFStore [122], and Neo4EMF [123]. CDO is a model repository and a runtime persistence framework which allows storage and access to EMF models and metamodels using many kinds of database back-ends. ModelBus is a web service application to manage an embedded subversion engine that implements the actual repository. EMFStore is a model repository for EMF which provides features for collaborative editing and versioning of models. Neo4EMF, based on the Neo4j model repository, is a scalable persistence layer for EMF models.

MDR [124] is a Java Meta-data Interface (JMI) implementation of MOF. Metamodels and models can be imported and exported from MDR using XML. Models are accessible through the JMI API, or programmatically using metamodel-specific access. Other tools use their own model repositories, such as the MetaEdit+ repository (MetaEdit+), JGraLab (MOLA), METADEPTH (Eclectic), the VMTS repository (VMTS), the GrGen.NET repository (GrGen.NET), MasterCraft (ModelMorf), and the Umple repository (Umple). Some model repositories can be extended to other frameworks using an API (supported by, e.g., AGE, Eclectic, VMTS).

Compatibility with well-known Standards, Specifications, and Languages (CS) [5,13,107] Ideally tools should provide support for well-known and legacy standards, languages and specifications, such as *CORBA Model Interchange* (CMI), to facilitate interoperability and migration between modeling tools. *XML Metadata Interchange* (XMI) [125] (supported in, e.g., TXL, Kermeta2, MDWorkbench, MagicDraw) is a model interchange language for serialization and exchange of models between modeling tools, and storage of models in data repositories using a structured textual XML file. The concrete syntax of XMI is verbose and not intended to be readable by humans. Thus, the OMG has also defined the *Human-Usable Textual Notation* (HUTN) [126], a standard supported in AToMPM, Epsilon, and Blu Age. ModTransf, ModelAnt, and Acceleo support the *Java Metadata Interface* (JMI) [127] standard, which allows different UML tools to interact with each other through APIs. In addition, tools such as AToM3, AToMPM, Blu Age, Henshin, and AGG support *Diagram Definition* (DD) specifications [128], which facilitate definition of the mappings between model elements and their graphical representations.

The OMG has standardized the expression of transformations used in Model-Driven Architecture (MDA), a subset of MDD, by introducing the *MOF Model to Text Transformation Language* (MOFM2T) [129] for M2T transformations, and the *Query/View/Transformation language* (QVT) [102], for M2M transformations. QVT is itself refined into three languages, namely QVTr (Sect. 4.1.1), QVTc (Sect. 4.1.1), and QVTo (Sect. 4.1.2). A number of tools, including Merlin, JQVT, JTL, PTL, ATL, and BOTL, support QVT-like notations. VMTS originally provided support for the QVT standard. The MOFM2T standard (implemented in Acceleo, MOFScript, and ModelAnt) is based on templates and OCL expressions and has language features, concrete and abstract syntax for specifying M2T transformations.

The *Common Warehouse Metamodel* (CWM) standard [130] (supported in, e.g., Tefkat and Kermeta2) eases the interchange of data warehouse and business intelligence meta-data between data warehouse tools, platforms and meta-data repositories in distributed heterogeneous environments. CWM suffers from capability limitations, so it is not very common in model transformation tools. The *Object Constraint Language* (OCL) [131] (used in, e.g., TGGInterpreter, MoTE, ATL, UMLX, ModelMorf) can be used to express relational constraints on the metamodel, and validation queries at the model level. QVTr uses OCL to specify templates, and when and where conditions in relations (implemented in, e.g., ModelMorf).

Reverse Engineering (RE) [17] Some tools can be used to aggregate, synthesize, and abstract information to, e.g., generate models from code, also known as reverse engineering. Examples of such tools include MDWorkbench, ModelAnt, MetaEdit+, Modelio, Umple, Blu Age, Fujaba, MagicDraw, SPARX, Together, and Acceleo. ModelAnt offers reverse engineering features for analyzing Java sources to reveal their packages, classes, attributes and methods, and for reverse engineering databases using the standard JDBC API, revealing their tables, columns, primary keys, foreign keys and relationships.

Round-trip Engineering (RT) [10] During software development, different models of systems at different levels of abstraction may exist. Some of these models can be generated or created using model transformation. Round-trip engineering is an effort to keep these models consistent by propagating changes from one model to other related models. More specifically, round-trip engineering can be considered as a three step task: (1) detecting changes to a model under consideration, (2) finding other models affected by the change, and (3) propagating the change to the affected models [132]. A number of techniques, including bi-directional transformation, incremental transformation, model synchronization, reverse engineering and traceability, can be used to implement round-trip engineering. Hettel et al. [133] have proposed a formal definition of round-trip engineering in the context of model transformation, categorizing the restrictions that are necessary to support consistency across transformations.

This facet is hard to support and only a few tools do so, including MetaEdit+, MagicDraw, Modelio, SPARX, Together, MoTE, GRoundTram, Blu Age, and Fujaba. The

primary round-trip feature supported by these tools is keeping generated code and models synchronized. Typically, this is performed by tracing changes in models and code, and propagating the changes using incremental model transformation and reverse engineering.

5.3 Transformation category

The *Transformation* category contains facets related to the model transformation language(s) supported by the tools. Table 5 summarize these facets and their attribute values.

Model Transformation Language Syntax (MTLX) [4,15,17,107,134] This facet provides information about the syntax of the model transformation language of the tool. Some tools, such as Tefkat, PTL, ModTransf, Echo, TXL, and VIATRA, provide a *textual syntax* for their language, while others (e.g., AToMPM, AGG, BOTL, TGGInterpreter, SPARX, Actifsource) provide *graphical syntax*. There are tools that have both graphical and textual syntax, such as VMTS, DSLTrans, GROOVE, JTL, UML-RSDS, and GRoundTram. In GROOVE, parts of the production system must be edited graphically (rules and graphs), and others textually (control, LTL/CTL properties and Prolog predicates).

Output (O) [4,5,17] The *Output* facet indicates the kind of output supported by a particular M2M or M2T tool. The output of M2M tools can be either *in-place* (destructive, modifying the original input model) or *out-of-place* (conservative, retaining the original while producing a new modified model). In-place transformations (supported in, e.g., UML-RSDS, QVTo-Eclipse, Kermeta2, GROOVE, VMTS, ATL, Fujaba) do not preserve the input, i.e., they directly modify the input model to obtain the output model [5,17]. In this case, the output model is created by directly creating, deleting, and updating elements of an existing input model. Model refinements, model refactoring, and model optimizations are examples of in-place transformations, where elements of the model that remain unchanged need not be copied to be part of the output. In some in-place systems, it is possible to emulate out-of-place transformations by copying the input model to the output model and then modifying it. In-place transformations are most suitable for endogenous transformations [17] and are normally implemented using tools based on graph rewriting, such as GROOVE, AToMPM, GRoundTram, and MO-MoT. Out-of-place transformations (supported in, e.g., Henshin, UMLX, Echo, PETE, ModelAnt, Eclectic) generate a new output model from scratch [5,17]. Out-of-place transformations are more suitable for handling exogenous transformations.

Source code, databases and textual artifacts are examples of outputs produced by M2T tools. TXL has been used to transform models in XMI form to Prolog facts, and to Structured Query Language (SQL) tables. MagicDraw can produce textual artifacts such as plain text, RTF, HTML, XML templates, and database schemas (without data). Umple can produce documentation in HTML and JavaDoc formats, as well as analysis reports. Many tools can produce source code as output. For example, MetaEdit+, Xtend, ModelAnt, VMTS, and Actifsource can all generate source code of various kinds. Acceleo generates JavaEE, C#, Python, Zope, PHP; Modelio generates Java, C++, C#, SQL; SPARX generates C++, C#, Java, Delphi, VB, SQL; VIATRA generates Java, C++; AGE, Henshin, Fujaba, Merlin, and ModTransf generate Java; MagicDraw generates Java, C#, C++ ; Together generates Java, J2EE, C++, C#; Blu Age generates JEE, .Net, JavaScript; Umple generates Java, C++, Ruby, PHP; and TXL can generate any programming language for which it has a grammar.

Cardinality (C) [5,12,17] The *Cardinality* facet classifies model transformations based on the number of input and output models that can be handled. Tools can support 1-to-1, 1-to-N, N-to-1, or N-to-N model transformations. 1-to-1 tools (e.g., UML-RSDS, MOLA, SPARX, Acceleo, TGGInterpreter, Fujaba, DSLTrans) are restricted to one input model and one output model. 1-to-N tools, such as UML-RSDS, VMTS, TGGInterpreter, PTL, Acceleo, ModelAnt, UMLX, AGG, Merlin, Mitra2, JQVT, can produce several output models from one input model, whereas in N-to-1 tools (e.g., UML-RSDS, TGGInterpreter, Mitra2, Tefkat, AGG, JTL, QVTR-XSLT) can handle several input models to generate a single output model. N-to-N tools (e.g., UML-RSDS, TGGInterpreter, Mitra2, ModTransf, ModelMorf, Umple, Merlin, AGG, SPARX, JTL, UMLX) allow one or more input model(s) to be transformed into one or more output model(s). In ModelMorf, relations among N models can be specified, but the user can only transform one model at a time to enforce the specified relations.

Rule Scheduling (RS) [4,12,134] Rule-based transformation systems vary widely in the mechanisms that determine the order in which individual rules are applied. The rule application mechanism can vary in four main respects: form, rule selection, rule iteration, and phasing [4]. Rule *form* refers to whether the order of rule application is implicit or explicit. In *implicit* scheduling, the transformation engine selects the execution order, and the user has no explicit control over the scheduling algorithm specified by the tool [4]. This mechanism is common in rewriting-based relational model transformation tools, such as Tefkat, JTL, PTL, PETE, and QVTR-XSLT. In the *explicit* style, the user has direct control over execution order of rules using explicit features of the transformation language, such as loops and conditionals. This style is dominant in the imperative tools, e.g., Xtend, QVTo-Eclipse, Umple, Melange, JAMDA, JQVT, Merlin, ModelAnt, Kermeta2, MagicDraw, and in the hybrid model transformation tools (e.g., Fujaba, VMTS), in which control flow is typically user-defined. There are various ways to specify control structure, for example using story diagrams

Table 5 Facets in the transformation category

Facet	Abbr.	Attributes		Percentage
MT Language Syntax (MTLX)	a	The tool has graphical syntax		48%
	b	The tool has textual syntax		63%
Output (O)	a	M2M	In-place/Destructive	58%
	b		Out-of-place/Conservative	75%
	c	M2T	Textual artifacts	40%
	d		Source code	40%
	e		Database artifacts	23%
Cardinality (C)	a	1-to-1		87%
	b	1-to-N		58%
	c	N-to-1		55%
	d	N-to-N		48%
Rule Scheduling (RS)	a	Form	Sequential/explicitly	68%
	b		Non-sequential/implicitly	45%
	c	Rule Selection	Explicit condition	58%
	d		Nondeterministic	37%
	e		Interactive	18%
	f	Rule Iteration	Recursion-oriented	42%
	g		Looping-oriented	38%
	h		Fixed-point-oriented	30%
	i	The tool supports phasing		22%
Rule Organization (RO)	a	The tool supports modularity		57%
	b	Reuse using inheritance		45%
	c	Reuse using logical composition		33%
Rule Application Control (RAC)	a	Deterministic		62%
	b	Nondeterministic/concurrent		33%
	c	Nondeterministic/single-point		23%
	d	Interactive		27%
Type (T)	a	Exogenous transformations		83%
	b	Endogenous transformations		70%
Direction (D)	a	Multidirectional transformations		5%
	b	Bidirectional transformations		22%
	c	Unidirectional transformations		100%
Verification (V)	a	Syntactic correctness		35%
	b	Termination		18%
	c	Semantic correctness		33%
	d	Completeness		8%
	e	Determinism/Uniqueness/Confluence		12%
	f	Robustness		13%
	g	Definedness		12%
Validation (VA)	a	The tool provides a testing environment		33%
	b	The tool provides a simulation environment		25%
Traceability (TR)	a	Automatic		47%
	b	User-defined		45%
Incremental Updates (IU)	a	The tool supports incremental updates		47%
Concurrent Transf. (CT)	a	The tool provides concurrent transformations		25%
Live/Active Transf. (LT)	a	The tool provides live/active transformations		5%
Transf. Suggestions (TS)	a	The tool supports transformation suggestions		7%
Changeability of MT (CH)	a	Access transformations		100%
	b	Add transformations		100%
	c	Update transformations		100%
	d	Delete transformations		100%

based on UML activity diagrams in VMTS, abstract state machines (ASMs) in VIATRA, state-charts and activity diagrams in Fujaba, rule priorities in AToM3, and transformation units in Henshin. Although the explicit form provides full control over transformation execution, it may be desirable to combine it with the implicit form for complex scenarios, since the developer's work increases with the complexity of the scenario. Examples of tools supporting the joint use of both implicit and explicit rule scheduling are ATL, Mitra2, GROOVE, Epsilon, Eclectic, UML-RSDS, and TXL.

Rule selection can be explicit condition, nondeterministic, or interactive [4]. In the *explicit condition* approach

(used in, e.g, PTL, UMLX, QVTo-Eclipse, MDWorkbench), an algorithm controls the order of application of rules. The order may be *nondeterministic* (e.g., used in AToMPM, DSLTrans, AGG, JTL), allowing for different executions of the same transformation on the same input model, possibly yielding different results. In *interactive* selection, the user can be involved in deciding how different transformation rules can be scheduled. Interactive selection is supported in AToMPM, Xtend, AToM3, Mitra2, and GrGen.NET. Template approaches usually offer user-defined scheduling with explicit calls to a template from within another one. For example, in Acceleo, rules (or individual templates) are only applied when called explicitly by name from another template, with the exception of the template that begins the generation. For a given name, several templates may apply, some with guard predicates or applicable to a more specific type. In that case, the engine invokes rules by checking the hierarchy to find the most specific template that satisfies its guard.

Rule iteration mechanisms include recursion, looping, fixed-point iteration (i.e., repeated application until rule application leaves the input unchanged), and combinations of these [4]. In MOMoT, rule scheduling is derived based on the quality of models, using a fixed-point iteration optimizing the quality of the output models.

Phasing, found in QVTo-Eclipse, AGE, PETE, ModelMorf, Tefkat, and ModTransf, organizes the process of transformation into a sequence of phases, each of which is a set of rules that has a specific purpose. The rules in a phase are applied until they complete before going on to the next phase. This can help users manage complex transformations, load target models safely, improve modularity, and provide control over the execution of transformation [135]. Kermeta2 does not provide built-in phasing; however, it allows explicit organization of rules into phases. Similarly, TXL transformations are often organized into phases by hand.

Rule Organization (RO) [4,12,15,134] This facet considers the organization and composition of transformation rules into rule sets and whole transformations [4]. *Modularity* mechanisms (supported in JTL, Acceleo, MOLA, GROOVE, GrGen.NET, Henshin, VIATRA, AGE, MOMoT, eMoflon, Mitra2, ModTransf, Melange) allow rules to be grouped into modules. Rule reuse mechanisms allow definition of new rules based on one or more existing rules. Reuse mechanisms can allow for *inheritance* between rules (in, e.g., MDWorkbench, AGE, UMLX, eMoflon, JQVT), and/or *composition* of rules (in, e.g., QVTR-XSLT, MoTE, UMLX, GROOVE, GrGen.NET, MOLA, MOMoT, XTL), avoiding duplication and helping with transformation maintenance. Tools which provide language composition and inheritance, such as AGG, Blu Age, Fujaba, Eclectic, Xtend, QVTo-Eclipse, and ModelMorf, can ease the development of model transformation. QVTo-Eclipse supports logical composition using disjunction and merging mappings. In QVTr, a rule can be composed of other rules by invoking them in its where clause (e.g., implemented in ModelMorf). TXL provides explicit rule abstraction using the notion of subrules.

Rule Application Control (RAC) [4,9,12,15,134] This facet is related to mechanisms determining where in a given input scope a rule should be applied. When more than one match exists for a rule, rule application control can be used to determine the application locations [4]. The control mechanism can be deterministic, nondeterministic or interactive [4]. *Nondeterministic* strategies can be either concurrent or single-point. If a rule can be applied at several matched locations at once, a nondeterministic mechanism with *concurrent* application (supported in, e.g., Tefkat, JTL, Echo, UMLX) can be used. In single-point application (supported in, e.g., TGGInterpreter, MOMoT, VIATRA), a rule is applied to one nondeterministically selected location. If the transformation is nondeterministic, the transformation result may not be unique. Using an *interactive* mechanism (supported in, e.g., Mitra2, AToMPM, AGG, eMoflon), the user decides to which location the rule should be applied. GROOVE provides a combination of rule application control strategies to find all locations to which a rule can be applied, and then allows the user to explore the best way to proceed by choosing to apply the rule either everywhere at once, at one location deterministically (i.e., the same one every time the same transformation is run), at one location randomly, or at a user-defined location manually.

Type (T) [4,5,12,17,107] Model transformations can be endogenous or exogenous, depending on the source and target metamodels [5,17]. In *endogenous* (also known as homogeneous or rephrasing) transformations, the input and output models conform to the same metamodel. Optimizations and refactorings of models are examples of endogenous transformations (supported in, e.g., GROOVE, AGG, BOTL, GRoundTram). *Exogenous* (also known as heterogeneous or translation) transformations manipulate input and output models conforming to different metamodels. Examples of exogenous transformations (supported in, e.g., Tefkat, Merlin, DSLTrans, Actifsource) are refining transformations, which refine a model to a more detailed form using code generation, reverse engineering, or platform migration. For example, the translation of a platform-independent UML model into a platform-specific Java model is exogenous [15].

Direction (D) [4,5,10,12,13,15,134,136–138] Transformations can be unidirectional, bidirectional or multidirectional. *Unidirectional* transformations can be executed only from input models conforming to a source metamodel to corresponding output models conforming to a target metamodel in the source-to-target direction. The QVTo tools, such as QVTo-Eclipse, are examples of unidirectional systems. In *bidirectional* tools, transformations can also be run in reverse or in multiple forward/backward directions between multiple

models. Bidirectional tools are most often relational. Examples of tools supporting bidirectional transformation include JTL, Echo, mediniQVT, ModelMorf, and UML-RSDS. Bidirectional transformations can be specified either as a pair of unidirectional transformations: a forward transformation (source-to-target) and a backward transformation (target-to-source), or using a relational approach, where every transformation relationship simultaneously describes a forward and a backward transformation. GRoundTram uses the pair of transformations approach, but automatically derives backward transformations so that users need not specify them. TGGs, BOTL, JTL, and QVTr are the examples of the relational approach. All of the TGGs tools, including eMoflon, TGGInterpreter, MoTE, and EMorF, support bidirectional model transformations. While ModTransf is designed to support bidirectional transformation, it has only partially been implemented. *Multidirectional* can be seen as supporting bidirectional transformations between a set of input and output models, which can facilitate traceability. QVTr and QVTc both support multidirectional rules, while the QVTo tools, such as QVTo-Eclipse, support only unidirectional. Echo has a prototype implementation of multidirectional transformations [139]. Bidirectional / multidirectional transformations can assist in checking and enforcing consistency between input and output models, for reverse and round-trip engineering, for software evolution, and for input-output model synchronization. Detailed reviews of bidirectional / multidirectional transformations can be found in several other papers, including Czarnecki et al. [140], Stevens [138], and Hidaka et al. [10].

Verification (V) [5,8,9,107,137,138,141] The verification facet addresses whether a tool provides support for formally verifying transformations with respect to properties such as correctness, consistency, completeness, termination, determinism/confluence, robustness, and definedness.

Syntactic correctness (supported in, e.g., UML-RSDS, JTL, PTL, Echo) deals with checking that well-formed models conforming to the source metamodel will always result in well-formed models conforming to the target metamodel [5]. Syntactic correctness is well understood and often automatically checked on output models by modeling tools. However, proving that a transformation only produces syntactically correct outputs for all possible inputs, is very difficult, especially for Turing-complete transformation languages [142].

Semantic correctness deals with verifying that certain properties of the source model are preserved in the target model(s) [5]. The properties can differ based on the goal of transformation. For example, when refactoring a behavioral model such as a state machine, the external behavior should be preserved. Properties can be defined using constraint languages (e.g., OCL, temporal logic) and verified using different verification techniques, such as model checking (e.g., [143]) and theorem proving (e.g., [144]). Henshin, MagicDraw, and Fujaba all support this attribute.

Completeness (supported in, e.g., UML-RSDS, VIATRA, MoTE, BOTL) can be considered from the point of view of source-complete, or of target-complete. Source-complete means that each element of the input model is transformed to a corresponding element of the output model. Target-complete means that each element of the output model is transformed from a corresponding element of the input model.

Termination (supported in, e.g., GROOVE, AGG, BOTL, GRoundTram) guarantees that, for given input model, a model transformation will eventually stop executing [145]. Most model transformation approaches are Turing-complete, and thus, termination is undecidable [146]. DSLTrans is limited to a Turing-incomplete set of language constructs and thus manages to guarantee termination of the transformation by construction. Other studies have proposed sufficient conditions for the termination of model transformations [147–149], and ATL, for example, can guarantee termination as long as lazy rules and called rules are not used [150]. *Determinism*, sometimes also called *confluence*, (supported in, e.g., UML-RSDS, GROOVE, AGG, BOTL, GRoundTram) is an important property that guarantees that different executions of the same transformation on the same input will always produce the same result.

Robustness considers whether unexpected errors of the transformation can be handled during execution. Tools such as PTL, Echo, Umple, GROOVE, VIATRA, Blu Age, and AGG all support robust transformations. Finally, *definedness* [146] captures whether a transformation definition is validly defined and applicable on every input model. Tools such as UML-RSDS, Echo, MoTE, Melange, and VIATRA check for definedness.

Validation (VA) [3,5,8,13,107,138] The behavior of model transformations can be validated using either testing or simulation. *Testing* executes a model transformation on test input models. Tests alone cannot fully verify that a model transformation behaves as expected on all possible input models. Nevertheless, testing with an adequate set of test inputs is typically an easy way to validate model transformations. The main drawback of testing is that it typically cannot fully verify a transformation, because exhaustive testing is time-consuming and often impossible. Tools such as Kermeta2, Modelio, Actifsource, VIATRA, MDWorkbench, and Fujaba provide a model transformation testing environment. For example, Fujaba uses graph-transformation-based JUnit tests. Epsilon uses a unit testing framework called EUnit for model management tasks. *Simulation* allows designers to validate the behavior of an executable output model by simulating its execution. Examples of tools that support simulation include MetaEdit+, Echo, AToMPM, GROOVE, AGG, BOTL, and VMTS. The simulator component in

GROOVE is a GUI-based tool allowing the construction, visual simulation, and model checking of rule systems. Xtend, Umple, MagicDraw, Together, Fujaba, SPARX, and GrGen.NET provide both simulation and testing environments. Formal verification and validation of model transformations has increasingly been a topic of research in recent years [142,143,151–157].

Traceability (TR) [4,5,8,10,12,15,107,134] Traceability, that is, the creation and maintenance of links between input model elements and their corresponding output elements, can be useful for analyzing transformations, performing impact analysis for manipulated model elements, determining the source/target of a transformation in model synchronization, and consistency checking between the input and output models. Traceability information is important and can be automatically generated by the model transformation tool, for example in PTL, QVTR-XSLT, mediniQVT, Umple, MDWorkbench, Together, GRoundTram, eMoflon, Mitra2, Merlin, EMorF, DSLTrans, and SPARX. On the other hand, traceability links can also be defined and generated by the user, as supported by UML-RSDS, PETE, Xtend, Kermeta2, Melange, GROOVE, Henshin, MOLA, Actifsource, GrGen.NET, AToM3, AToMPM, VMTS, AGE, ATL, Fujaba, and GReAT. MoTE automatically creates and maintains a traceability model between the models manipulated by a transformation. The generated trace model is used by MoTE to check or maintain consistency between input and output models as changes are applied to them. Traceability for MOFScript has been designed, but not yet completely implemented. Some tools (e.g., Tefkat, JTL, Modelio, MoTE, VIATRA) support both automatically generated and user specified traceability information. Traces are automatically generated if the user does not specify any. Winkler et al. [158] surveyed the challenges of supporting traceability in requirements engineering and MDD.

Incremental/Persistent Updates (IU) [4,5,9,12,14,15,17,107,134,136,141]: This facet refers to the way in which changes to the input model of the transformation are propagated to the output model and vice-versa. In systems that do not support incrementality (e.g., Henshin, Kermeta2, BOTL and TXL), the complete output model must be regenerated if the input model is updated. Czarnecki and Helsen [4] identify three types of incrementality: target, source, and user-edit-preserving in output. In *target incremental* transformations, output models are incrementally updated based on changes to the input models. *Source incremental* transformations aim at minimizing the number of input elements that need to be rechecked by a transformation when the input is changed. In *user edit-preserving* incrementality, changes made by users to the output model are preserved even when the output is regenerated. Incremental model transformation can be implemented using in-place transformations [159].

Tools such as JTL, VIATRA, Actifsource, Acceleo, Fujaba, VMTS, EMorF, AToMPM, Epsilon (EGL), and MoTE all provide incremental transformation. Tefkat can support incremental transformations if the trace of the transformation or output models are partially populated. Xpand does not support incrementality, since there is no clear mapping from changes to the input model to output text fragments [160]. EMorF supports incremental transformation in both directions. mediniQVT uses traces to enable incremental updates, preserving manual changes to the output model even when the entire transformation is re-run.

Concurrent Transformations (CT) Concurrent transformation refers to running two or more transformations at the same time. This is useful for supporting high-performance transformations in large-scale projects. Some attributes, such as orchestration, can prevent transformations from concurrently manipulating the same model elements, if the transformations share the same model in memory. Tools that support concurrent transformations include Xtend, MetaEdit+, QVTo-Eclipse, Umple, MDWorkbench, MagicDraw, Together, AToMPM, AGG, BOTL, Henshin, MoTE, GrGen.NET and Blu Age. In AToMPM, transformations can be run concurrently on two different models. Previous versions of VIATRA (before 2010) supported concurrent transformations.

Live/active Transformations (LT) This facet addresses the ability to automatically run model transformations in the background as daemons triggered by changes in the underlying models [161]. This allows a transformation to be executed automatically as soon as a transaction on the model has completed. Examples of tools supporting this facet are VIATRA, Together, and Blu Age.

Transformation Suggestions (TS) [5] This facet (called *content assist* in IDEs) indicates tools that offer mechanisms for determining which model transformations can be appropriately applied in a given context [5]. For example, the tool might not only be able to apply refactoring transformations, but might also suggest the contexts where a particular refactoring can be applied [5]. Normally, the developer is expected to determine and define the most effective contexts in which to use a transformation. However, this facet can increase usability for beginners and nonexpert users by helping them identify appropriate contexts. This facet is supported by AGG, MOMoT, VIATRA, and Blu Age.

Changeability of Model Transformation (CH) [5] This facet is related to creating, reading, updating, and deleting (CRUD) model transformations and thus is the most trivial and basic facet that model transformation tools should support. However, there are special cases where supporting CRUD is not as trivial as expected. For example, when tools

provide predefined transformations, the ability to update them may not be provided.

5.4 User experience category

User Experience addresses facets pertinent to the user-friendliness of the tool, and the quantity and quality of available supporting resources. The richer these facets, the more user support the tool provides. Table 6 summarizes these facets and their attribute values.

User Interface (UIN) [8,9] This facet describes whether the tool provides graphical, command-line/textual, or both graphical and command-line user interfaces. Some tools, such as Umple and AToMPM, provide access to the tool through a custom web application. PTL, QVTo-Eclipse, MOLA, and Actifsource are examples of tools with graphical user interfaces. Examples of tools supporting command-line user interfaces are TXL, Echo, VIATRA, and eMoflon. The importance of a tool's user interface and its effect on the user experience has been extensively studied in other software domains [162–164].

Workspace and Project Management (WPM) This facet indicates whether the tool provides facilities to support easy organization and management of user artifacts such as projects and files. A majority of tools, including Tefkat, PTL, Echo, Xtend, MetaEdit+, QVTo-Eclipse, Kermeta2, Modelio, Umple, MDWorkbench, VMTS, AGE, MagicDraw, and Melange provide such support.

Syntax Editor (SYE) [8] This facet catalogs syntax-directed editing facilities such as syntax highlighting, auto-formatting, code completion, code navigation, and code folding, that allow users to work more easily with the tool. Some tools adopt facilities already provided by other tools. For example, Echo inherited editing facilities from QVTs Eclipse editor, and Mitra2 inherited its editor from Xtext. Examples of tools that provide full support for this facet are Echo, Xtend, Kermeta2, Modelio, MDWorkbench, Melange, MagicDraw, SPARX, Together, MOMoT, VIATRA, ATL, Blu Age, Acceleo, and Actifsource. TXL is supported by a custom Eclipse plugin.

Semantic Editor (SE) [8] This facet addresses support for semantics-sensitive editing, including refactoring, error and warning detection, quick fixes, debugging, reference resolution, automatic build systems, and profiling, to help the user trace and correct failure(s). Examples of tools that provide full support for this facet are Xtend, Modelio, Together, Blu Age, and Actifsource.

Level of Automation (LA) [5,10,11,136,138] This facet considers the level of automation in the execution of model transformations that exhibit nondeterminism due to, e.g., choices during rule application or scheduling. Resolution of nondeterminism can be either completely automated, done manually, or use a certain amount of user intervention (semi-automated). While manual approaches give the user a maximum of control, they also often require a deep understanding of how the transformation tool works. In general, semi-automatic and manual tools can therefore slow devel-

Table 6 Facets in user experience category

Facet	Abbr.	Attributes	Percentage
User Interface (UIN)	a	Graphical	87%
	b	Command-line	35%
Workspace&Proj. Mgmt. (WPM)	a	The tool provides workspace and project mgmt.	77%
Syntax Editor (SYE)	a	The tool has syntax highlighting	62%
	b	The tool has auto formatting	40%
	c	The tool has code completion	53%
	d	The tool has code navigation	43%
	e	The tool has folding	38%
Semantic Editor (SE)	a	The tool has refactoring	27%
	b	The tool has error and warning detection	82%
	c	The tool has quick fixes	33%
	d	The tool has a debugger	67%
	e	The tool has reference resolution	23%
	f	The tool has automatic build systems	23%
	g	The tool has a profiler	20%
Level of Automation (LA)	a	Manual	3%
	b	Semi-automatic	87%
	c	Automatic	10%
Automatic Reporting (AR)	a	The tool generates reports/documentation	22%
Programming Style (PS)	a	The tool is similar to programming languages	60%
	b	The tool is similar to scripting languages	20%
	c	The tool is uses mathematical/algebraic style	40%
	d	The tool is logic-based	7%

opment compared to fully automated ones. Only MetaEdit+, Modelio, eMoflon, VIATRA, MagicDraw, and Blu Age provide fully automated transformation.

Automatic Reporting (AR) This facet considers tool support for automatic report and documentation generation. For example, Kermeta2 can automatically create JavaDoc-like documentation, Kermeta can generate simple class diagrams for transformation programs, and ModelAnt generates transformation documentation in both RTF and WIKI formats.

Programming Style (PS) This facet captures the similarity of the programming style encouraged by the transformation language with established styles. Some transformation tools are based on traditional programming languages (e.g., Mitra2; Kermeta2 is an OO language; Xtend is based on functional programming and OO languages; QVTo-Eclipse is similar to procedural programming languages), or scripting languages (e.g., Modelio; Tefkat has a syntax similar to SQL; ModelAnt uses scripting over an OO domain/wrapper of the model). Languages based on well-known, established styles may be easier to learn for most users. Conversely, languages with less common notations and styles such as tools based on concepts from algebra (such as AToM3; also, GRoundTram is based on graph query algebra UnCAL) or logic (such as Tefkat; also, PTL and PETE are based on Prolog) may be harder to learn for the average user.

5.5 Collaboration support category

The Collaboration Support category groups facets required to build support for migration between tools, extend tools with new features, and use tools for large and complex projects (Table 7).

Teamwork Support (TSU) This facet indicates whether the tool provides support for the collaboration of several people on the same project (indicated by 'multiuser') or for the handling of multiple projects (indicated by 'multiproject'). MDD involves teamwork by nature, and models are often developed by independent teams of designers. Thus, team members may need to discuss the same models or collaborate on shared models simultaneously. Xtend, MetaEdit+, Kermeta2, Modelio, Umple, SPARX, MDWorkbench, MagicDraw, AToMPM, Eclectic, Blu Age, and Actifsource all provide facilities for collaboration and multiuser projects.

Re-usability Techniques (RUT) [5,107] This facet refers to techniques that enable reuse of test sequences, rules, functions, procedures, patterns, and transformations in order to boost scalability, efficiency, and maintainability of transformations. Composition, decomposition, higher-order transformations (HOTs), orchestration, and generics are different ways to provide reuse of a transformation or its individual rules and functions. *Composition* allows reuse of existing transformations and rules in new (and possibly more complex) transformations. Examples of tools that support composition are UML-RSDS, Tefkat, UMLX, Mitra2, PTL, ModTransf, ModelMorf, and PETE. *Decomposition* supports splitting complex transformations into smaller, more reusable components. Examples of tools that support decomposition are VIATRA, Tefkat, ModTransf, MetaEdit+, Together, PETE, TXL, and Eclectic. *Higher-Order Transformations* (HOTs) consider everything as a model, and allow transformations themselves to be reused as input or output models [17]. In the QVT languages, a model transformation is itself a model, which facilitates the construction of HOTs. Examples of tools that support HOTs include Henshin, MOLA, VIATRA, ATL, Epsilon, and Kermeta2. Several HOTs have been written in Kermeta2, and HOTs has been used to facilitate M2T transformations in TXL. *Orchestration* can be used to reuse transformations in the large (i.e., using whole transformations as components) [165]. Examples of tools that support orchestration are ModTransf, Epsilon, MetaEdit+, Xtend, QVTo-Eclipse, Together, BOTL, Fujaba, AToMPM, GrGen.NET, Blu Age, MOMoT, and Eclectic. For example, Epsilon supports orchestration through ANT and EGX for templates. *Generic* transformations parameterize the transformation with the metamodel type and make the transformation logic independent of the metamodel [165]. Examples of tools that support generic transformations are Xtend, MetaEdit+

Table 7 Facets in collaboration support category

Facet	Abbr.	Attributes	Percentage
Teamwork Support (TSU)	a	The tool is multi-user	32%
	b	The tool is multi-project	32%
Re-usability Technique (RUT)	a	Composition	57%
	b	Orchestration	30%
	c	Decomposition	27%
	d	Generics	18%
	e	Higher-order transformations (HOT)	17%
Interoperability (IN)	a	The tool has a version control system (VCS)	33%
	b	The tool provides import/export of models/ metamodels developed using other tools	82%
Extensibility (E)	a	The tool supports extensibility	65%

Melange, MagicDraw, Together, Fujaba, GrGen.NET, Blu Age, VIATRA, ModelAnt, and Kermeta2. ModelAnt provides an OO wrapper around the model, which allows reuse of the methods, while changing only the templates for a specific generation task.

Interoperability (IN) [5,8,13,107] This facet highlights support for interoperability, such as support for version control systems (VCSs), and import/export mechanisms for models/metamodels developed using other tools. Generally, a single MDD tool does not support all of the required tasks in the model transformation process. Thus, integration and exchange of models, metamodels and other information with other tools is important. Efficient import/export facilities (e.g., supported in Merlin, GRoundTram, Eclectic, Henshin) allow the tool to cooperate with other external tools.

In large projects, it is necessary to handle concurrent modifications and versions, to detect conflicting modifications, and merge modifications made by multiple developers. Standard VCSs (e.g., supported by MDWorkbench, EA, Together) can automatically detect and resolve these issues in order to maintain a uniform version. VCSs can also be used to assist in re-usability. In the MDD-context, VCSs should provide both textual versioning and graphical representation. A detailed review of VCS technologies can be found in [166].

Extensibility (E) [5,8,13,107] Extensibility considers support for integrating new technical solutions and functionalities into the tool, and for modifying existing ones. JTL, MDWorkbench, TGGInterpreter, AGE, Eclectic, Fujaba, GROOVE, and ModTransf all include facilities for extending the tool. MetaEdit+ has a SOAP-based API that allows users to integrate other tools, or to add their own new functionalities. GrGen.NET supports internal extension with user programming of new features, and external extension with support for external libraries.

5.6 Runtime category

This category is related to the runtime environment and execution of the tool. Table 8 summarizes these facets and their attribute values.

Operating System (OS) Platform [8] The facet describes the OSs for which the tool is available. The majority of tools provide support for all three major OSs, including Linux, MacOS, and Windows. This may be due to the fact that the majority of them are developed using the platform-independent Java programming language.

Execution Environment (EM) [8] This facet captures whether the tool is part of an integrated development environment (IDE) or is a standalone application. Some tools, such as QVTR-XSLT, which uses MagicDraw, are dependent on other IDEs. Only a few tools support their own IDE, for example, VMTS has its own IDE called VMTS Studio, and GReAT uses the Generic Modeling Environment (GME) IDE. The extensibility of the Eclipse platform have made it a popular choice for model transformation tools (e.g., Xtend, eMoflon, Henshin). Examples of tools with a standalone environment are UML-RSDS, AToMPM, GRoundTram, Kermeta2, AGE, and Acceleo. Kermeta2 provides a way to compile Kermeta programs to be run as a plain Java application, and GROOVE can be run standalone in the JVM.

Execution Model (EXM) [8,17] Once defined and syntactically and semantically validated, model transformations must be executed. Techniques to implement transformation execution include: *code generation*, which generates executable code from a higher-level specification of the transformation; *interpretation*, which parses and executes the transformation at runtime; and a *hybrid* of interpretation and code generation (supported by, e.g., MOMoT, eMoflon, GrGen.NET, VMTS, UMLX, MoTE). In MoTE, TGGs are transformed into models of story diagrams and then interpreted to perform the transformation. eMoflon adopts a hybrid approach in that its unidirectional transformations and TGG rules are compiler-based, whereas the TGG execution engine itself is interpreter-based. In general, GrGen.Net is compiler-oriented, but the rule application language is interpreted (unless used in embedded rules). In JQVT, Java code is generated from a QVTr transformation that does not need to re-interpret the transformation rule. Thus, JQVT can produce faster transformations, which can be embedded into a

Table 8 Facets in the runtime category

Facet	Abbr.	Attributes	Percentage
Operating System (OS)	a	The tool runs on Windows	100%
	b	The tool runs on Linux/Unix	92%
	c	The tool runs on MacOS	85%
Execution Env. (EM)	a	The tool is a plugin for Eclipse	73%
	b	The tool is integrated/depends on other IDE	20%
	c	No IDE support	18%
	d	The tool has a standalone app	52%
Execution Model (EXM)	a	The tool is interpreter-based	48%
	b	The tool is compiler/code generator-based	70%
External Dep. (ED)	a	The tool has no external dependencies	19%
	b	The tool has external dependencies	89%

Java application more easily than traditional QVT scripts. TXL provides both interpretive and compiler-based modes.

External Dependencies (ED) The facet indicates whether the tool depends on other tools. Some tools are standalone, providing a transformation IDE, but depend on other tools for execution. Several of the assessed tools have dependencies on other tools. For example, Melange (which uses Xtext and the Kermeta3 Action Language (K3AL)), Xtend (which depends on Google Guava), AToMPM (which uses Python-igraph and Chrome), Mitra2 (which uses Xtext), BOTL (which needs ArgoUML), eMoflon (which needs SPARX and ANTLR), MOTE (which uses MDELab Story Diagrams), MOMoT (uses MOEA framework and Henshin), Blu Age (needs MagicDraw), MOLA (uses the METAclipse tool), MoTMoT (which needs AndroMDA and Maven), and VIATRA (which uses IncQuery and Xtend). Dependencies on other external tools can complicate installation and use of a tool.

6 Discussion

In this section, we discuss the findings and implications of our study. First, we begin with an overview of the surveyed tools, identify discontinued tools, and list factors that may have contributed to this discontinuation. Second, we discuss which of our facets are not well supported by the tools in general and consider how the underlying transformation approach may affects the support of facets. Finally, we examine the current status of model transformation tools, along with the popular standards, tools and languages.

6.1 Overview of tools

The 60 surveyed tools were developed between 1990 and the present. More than half of the tools are publicly available as open source. 77% of the tools in our study were developed in European countries, in comparison with 13% from North America, 7% from Asia, and 3% from Australia. 31 tools provide M2T transformation, using either template (19), visitor (5), or hybrid (7) transformation approaches. 56 tools provide M2M transformation, with imperative being the most popular transformation approach (17), followed by graph-based (15), hybrid (13), and relational (11). 27% of the tools have been discontinued, and 5% have been replaced by successors.

Figure 3a shows the trend of new, discontinued and replaced tools. The number of new tools has first increased and then decreased rather dramatically. The number grew rapidly until 2006, and then gradually decreased until 2010. Since 2010, the number of new tools has been more stable with only a slight decrease. In contrast to the number of new tools, the number of discontinued and replaced tools increased between 2007 and 2010, and is now more stable with a slight decrease. Despite the decrease in the number of new tools, the overall number of active tools (i.e., those actively maintained and updated) has been slightly increasing. As shown in Fig. 3b, hybrid and graph-based tools account for most of this increase, while the number of imperative and relational tools is now almost constant. Overall, the trend demonstrates a very high interest in model transformation tools between 2004 and 2007, followed by a drop and then a plateau.

Figure 4 presents the longevity (L) of each of the tools according to their first and latest release. The data show that 24 tools had a lifetime of less than 5 years. 20 tools lasted between 5 and 10 years, and only 16 of tools have survived more than 10 years. The average and median longevity of tools is 7.5 and 6 years, respectively. TXL and Metaedit+ currently have the highest longevity, and the shortest-lived active tool is currently Melange (because it is also the newest).

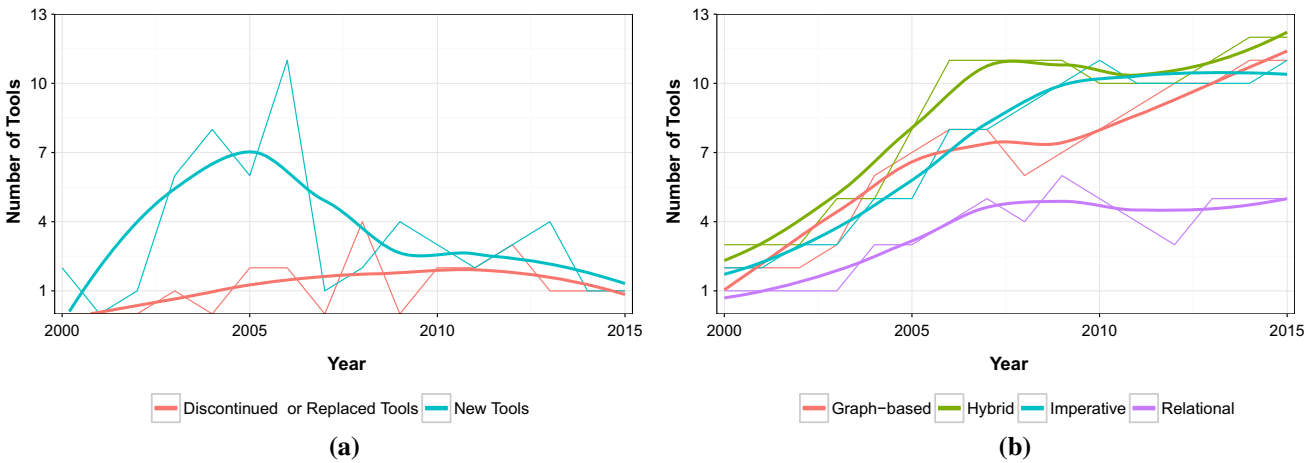

Fig. 3 **a** Number of new and discontinued. tools. **b** Number of active M2M tools, by approach. Trend of active tools, their introduction and discontinuation, from 2000–2015

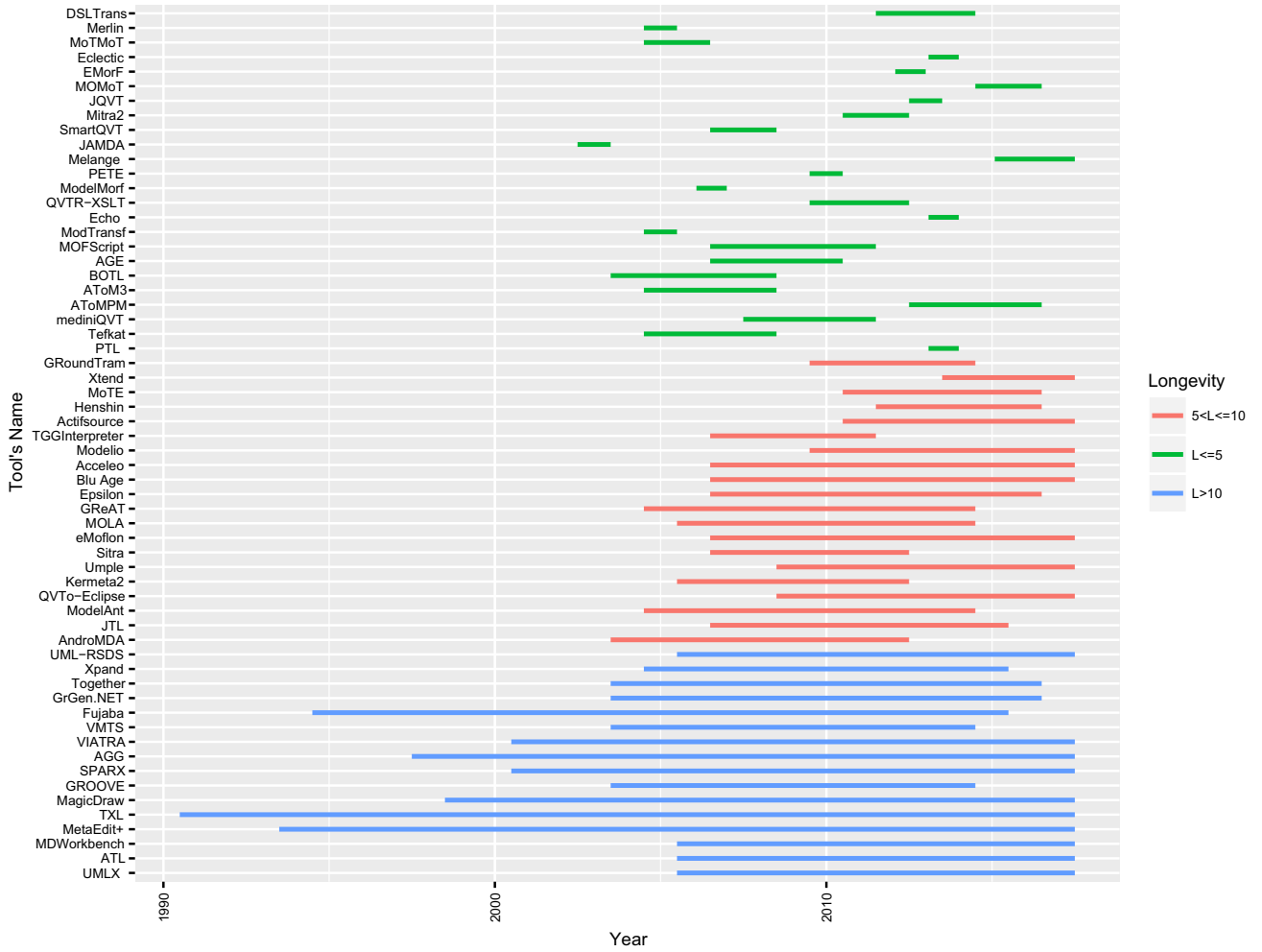

Fig. 4 Longevity of the assessed tools (years)

6.2 Analysis of discontinued tools

27% of surveyed tools have been discontinued, and a question to be considered is "why?" In the following, we consider in detail the observed common factors among them. These commonalities are grouped into four classes: *underlying transformation approach*, *standards compliance*, *number of developers*, and *supported facets*.

6.2.1 Underlying transformation approach

The underlying approach can contribute to the development process, the supported facets, and even the longevity of a tool. Our study shows that more than half (55%) of the relational tools have been discontinued. This percentage is notable compared to the discontinued tools using other approaches (i.e., 29, 20, and 7%, respectively, for imperative, graph-based, and hybrid tools). Moreover, with the exception of TXL, UML RSDS and JTL, none of the relational tools that are currently being maintained have been updated in the last four years, possibly an early indication of their upcoming abandonment. There are two major reasons why relational tools may not have enjoyed more long-term support: (1) due to the inherent complexity of the operations used in relational approaches, they may require more effort to maintain, and (2) the implicit nature of relational approaches may make them more difficult for nonexpert users to understand, leading to a smaller or less committed user base. Our findings present two potential topics for future research: an investigation into the reasons why such a large number of relational tools are no longer supported, and the introduction of new relational approaches that overcome these reasons. The second topic could also be expanded to include an examination of how existing relational tools could be combined with imperative approaches to make them simpler for users to understand and work with. The continued success of hybrid tools, such as ATL and VMTS, may be an indicator that this approach can be successful.

6.2.2 Standards compliance

The OMG has proposed a set of standards and guidelines such as Ecore, EMOF, XMI, and QVT, discussed in Sect. 5.2, as a way to standardize the development process for model transformation tools. Developers have tried to make their tools more interoperable by using these common standards. However, existing attempts have shown that some of the standards are complicated and time-consuming to align with, and challenging to understand. The complexity and ambiguity of these standards, e.g., QVT (i.e., QVTo, QVTr, QVTc), has been discussed in several studies [13]. In this context, our study indicates that four of the nine tools following the QVT standard are no longer maintained. After a 10-year wait, a new release of UMLX was made available in 2017. Echo partially implements QVT, but has not had an update for the last 3 years. There are just three active QVT-based tools; QVTO-Eclipse, which implements QVTo, and the industrial tools Together and MagicDraw. Mesa [13] shows that it is not clear how best to implement a QVT engine: using some kind of virtual machine implementation (as in, e.g., MOMENT, QVTo-Eclipse) or a direct implementation (e.g., mediniQVT, ModelMorf, SmartQVT). We observed a similar situation for EMOF, where more than 53% of the discontinued tools use EMOF as their metamodeling language, compared to only 14% of active tools.

It is important to note that the complexity of the standard is not the only factor in the decision to discontinue the support for a tool; the selection of the appropriate standard for the application domain also plays an important role. In particular, aligning a tool with OMG standards is time-consuming and likely is not a reasonable option for a short-term research project.

6.2.3 Number of developers

An old African proverb says: *"If you want to go fast, go alone. If you want to go far, go together"*. The proverb seems applicable to software development, in that teamwork is essential in developing successful tools. This claim is supported by our study. By estimating the number of developers using publicly available repositories of discontinued and mature tools, we noted that a remarkable number of the discontinued tools were developed by small groups of researchers (e.g., JAMDA, JQVT, Merlin). Many of these discontinued tools had early successes; however, support for them ended after a short time likely due to completion of graduate degrees, career changes, or switches to other projects. In contrast, we observed that the longest lived and most mature tools were developed by larger groups of researchers in research projects (e.g., VIATRA, Henshin) or professional teams in industry (e.g., Together, MetaEdit+, SPARX, MDWorkbench, MagicDraw).

6.2.4 Supported facets

We compared the support of facets between the discontinued and active tools to understand whether there are any noticable differences between them. Figure 5 shows the facets whose support by discontinued and active tools differs by more than 25%. The main differences are related to semantic and syntax services, technical support, validation and verification, and support for collaboration, indicating that support and scalability are critical to a tool's long-term success.

6.3 Rare facets

In this section, we discuss useful, and often crucial, facets of model transformation tools that are generally not well addressed by the surveyed tools. To do so, we counted the number of tools supporting each facet and then considered a facet to be rare if less than 50% of the tools provide support for it. The detailed analysis shows that the tools mainly focus on how to express model transformations (which is not surprising), but pay much less attention to other crucial facets, such as verification and validation. We observed that four kinds of problems are associated with this lack of support, including (1) the relative lack of support for facets crucial to *model management* tasks, such as synchronization, evolution, version control (supported by 33% of the tools), and traceability (supported by 47% of the tools); (2) the lack of support for the *user experience* facets, which affect the usability and learning curve of tools; (3) the lack of support for facets that facilitate *quality assurance*; and (4) the lack of *technical support*, which increases the risk and cost of using the tools in practical and mission-critical projects. Providing support for these rare facets is a necessary area of research, presenting several potential research topics.

6.3.1 Model management

Input and output models can be modified following the application of model transformations. Due to the iterative nature of software development, these modifications can be applied at all stages of development, from requirements analysis to ongoing maintenance. For example, it is possible that a model at a high level of abstraction, such as an architecture model, may be changed, and the changes should be detected and propagated to other dependant models. Without appropriate model management techniques, these changes can cause inconsistencies. It is, in this context, not sufficient to implement model transformation tools that simply transform an input model to an output model; richer support for transformed models over the software life cycle is required [167,168]. More concretely, support for model management, such as model comparison, model query (supported by 28% of the tools), incremental update (supported by 47%

Fig. 5 Facets with noticably different levels of support in discontinued versus active tools

of the tools), as well as live (supported by 5% of the tools), round-trip (supported by 15% of the tools), and bidirectional transformation (supported by 22% of the tools), is important. Model query and comparison are specifically important because they support versioning at the model level. Moreover, model comparison and automatic transformation are useful for supporting testing activities. Only a few of the tools in our study support model comparison and model query.

The *incremental* facet is important to model management as it means that the entire model transformation need not be rerun when only some of the model elements have changed. This optimization helps increase the efficiency of transformations of large and complex models. Live transformation is useful when analyzing the impact of a change in a model on other related models and also enables model transformations to be run in the background, providing better response time when dealing with large and complex models. Concurrent transformation (supported by 25% of the tools) can also help improve performance. Finally, bidirectional transformations and round-trip engineering can improve support for model management. Formulating and supporting these two facets could provide a precise and systematic solution to manage inconsistencies between models.

6.3.2 User experience

A good user experience is necessary for the success of software tools [169]. However, the user experience aspects of many model transformation tools have not progressed at the same rate as that of other tools; for example, our study shows that the majority of tools do not provide support for even such rudimentary user experience issues as syntax/semantics assistance and report/documentation generation (supported by 22% of the tools).

To determine whether user experience has been a focus for researchers in model transformation, we examined the topics listed in the calls for papers for the MODELS conference over the past five years (2011 to 2016). Surprisingly, we discovered that user experience has never been listed as a topic of interest. In terms of actual papers published over the same period, about 16 (e.g., [108,170–172]) of 230 (6%) were related to user experience of modeling tools. This suggests that the modeling community should pay more attention to user experience issues. Further, new generic and efficient frameworks should be introduced that support graphical and textual editors without dependence on a particular standard or plugin. Finally, the modeling community should collect and share their experiences with tools through publishing up-to-date tutorials, examples, and experience papers.

6.3.3 Quality assurance

Undetected errors and inconsistencies at different stages of model transformation can cause faults in the resulting models or source code, and these issues can cause further problems in dependent models. The propagation of faults can make their localization and repair an expensive and time-consuming task. This is even more important when model transformation is applied in the context of safety-critical systems. This issue can be addressed by providing adequate support for verification and validation tasks in model transformation tools,

something that currently does not receive the required level of attention.

The majority tools either do not support verification and validation, or provide only cursory support. While no tools provide full support for verification, validation is better addressed, with 20 tools (33%) supporting transformation testing. While many tools exist to verify and validate software, verifying model transformations is different from traditional program verification [173], and formal verification cannot always be immediately applied in model-based development. Low-level mathematical representations, such as Kripke structures or finite transition systems, are often the input for verification tools, whereas modeling concepts are based on high-level abstractions that independently specify relevant system aspects. The easiest way to apply existing techniques is to bridge the gap between the high- and low-level representations using a model transformation. However, these model transformations must themselves be verified first, which exacerbates the problem. It is evident that there is a need for more focused and dedicated methods to support verification and validation of model transformations.

6.3.4 Technical support

Generally, the ultimate goal of tool developers is to attract new users from both the research community and industry. Industry partners can provide real problems and insight to help developers transfer their approaches into practice, while collaboration among researchers allows the exploration of new ideas for addressing research challenges. Attracting early users of both kinds benefits both tool makers and their potential users. One of the most important ways of achieving this goal is by providing adequate technical support for developed tools. Proper technical support is even more important for model transformation tools since, because they often suffer from inadequate documentation and poor user experience. Specifically when dealing with industrial adoption of research tools, proper technical support helps assure industry partners that there is a long-term plan in place for support of the tool. Our study shows that only eight tools provide full technical support for their tools, whereas 22 tools provide partial support. As regards supporting resources, only 20 tools provide full support including documentation, examples, a wiki page and an online forum. The lack of supporting resources has been noted in other work [108].

It can be argued that open-source tools do not need professional technical support, because such support can be provided by the community of volunteer developers. To some extent, that may be true. However, at least limited support in terms of forums, wikis, bug tracking, and a review process are essential for active open-source communities, and these are not provided for many of the existing model transformation tools. Among the tools that do provide technical support, many leverage the Eclipse modeling community. The infrastructures offered by open-source foundations such as Eclipse and Apache can be very beneficial to the success of the tools. For example, many Eclipse modeling projects have public forums associated with them in which users can ask questions, report problems, and share solutions.

6.4 Impact of the underlying transformation approaches

As discussed in Sect. 6.2.1, the transformation approach adopted by tools may affect their success or failure. It is also interesting to consider how the adopted approach may affect the tools' support for facets. In following, we discuss the most important such observations.

Relational and graph-based tools support bi/multidirectional transformations more than other tools. This facet is difficult to support in imperative or direct manipulation approaches, as they define the transformation in an explicit, step-by-step way [15]. QVTr and QVTc can support multidirectional transformations, while QVTo supports only unidirectional ones. QVTr specifications, in tools such as ModelMorf, define the conditions under which a transformation can be bidirectional.

Verification, correctness, and consistency are supported by graph-based tools more than other tools. The mathematical basis of graph transformation tools encourages the development of formal analyses and approaches with theoretical guarantees (such as termination or confluence). However, scaling these analyses to industrial applications still is a challenge [174].

Reverse engineering, round-trip, and incremental transformations are mainly supported by template-based M2T tools. This is due the fact that code generation is a critical part of M2T tools and that the need for these three facets typically arises in the context of code generation.

Traceability is supported by graph-based tools less than other tools. Graph-based tools are in-place tools by their nature, as they are based on graph rewriting. This makes it difficult to maintain traceability links between input and output models, since the input model is often destroyed during the transformation.

Conflict resolution and interaction mechanisms for rule scheduling are more common in graph-based tools. Graph constructs can randomize rule scheduling, so the default mechanism for rule selection is nondeterministic. However, some graph-based tools, e.g., Henshin and GRoundTram, also support deterministic scheduling. Nondeterminism can cause conflicts between rules, leading to different, unexpected results. Hence, graph-based tools provide *conflict resolution* and *user interaction* to deal with nondeterministic choices. AGG and Henshin use a critical pair analysis [175] to detect conflicts between rules and also establish

sufficient conditions for the termination of graph transformations.

6.5 Standards, tools and languages

We observed that EMF/Ecore, XMI, and OCL are used by more than 60% of the surveyed tools. This observation has several implications. First, these standards greatly facilitate the development of interoperable tools. Second, teaching these languages and standards can be beneficial for new researchers and students, as they provide a basis for working with many of the existing tools. We could even argue that these languages should be part of every modeling course. Third, these tools can benefit more easily from existing and ongoing research on these languages and standards. For example, tools using EMF can adopt existing software and libraries for EMF model query and comparison (e.g., [176,177]) to support related features.

We also observed that many of the tools have been developed as Eclipse plugins. One of the reasons for the popularity of Eclipse as a host environment for model transformation is that it provides a wide range of tools supporting various aspects of modeling and MDE. As discussed in Sect. 6.3.2, this can allow tool developers to focus on the core functionality of their tools, while reusing existing Eclipse services to provide other tasks such as work-space management and syntax assistance.

7 Related work

To our knowledge, there has been no other similarly comprehensive survey and comparison of model transformation tools. However, several other surveys have assessed the capabilities of a subset of available MDD tools. Czarnecki et al. [4] have discussed the main features that can be used to compare model transformation approaches (e.g., features related to transformation rules, their execution, and manipulated metamodels). For example, their study classified available model transformation approaches as M2M and M2T. Although their work provides a useful hierarchical classification of model transformation approaches, it focusses on transformation rules, which is just one aspect of model transformations. Mens et al. [5] classified model transformation tools based on several factors, such as quality requirements and correctness verification, using a multidimensional approach. Another interesting study has been conducted by Jakumeit et al. [8], who surveyed 13 model and graph transformation tools from the 2015 Transformation Tool Contest (TTC). They compared the tools based on different criteria, including suitability of the tool and expected input. Jilani et al. [136] surveyed and classified 14 transformation approaches based on features such as availability of the tool, understandability of the transformation syntax, and whether the transformation approach is bidirectional or unidirectional.

Other surveys have focused on evaluating bidirectional transformation tools and approaches. Eramo et al. [107] proposed a taxonomy based on features pertaining to bidirectional transformations divided into three categories, *General Requirements* (e.g., complexity of transformations), *Functional Requirements* (e.g., traceability), and *Nonfunctional Requirements* (e.g., scalability). Similarly, Hidaka et al. [10] proposed a feature-based approach to compare different bidirectional model transformation approaches according to four major categories: Technical Space supported by the tool (i.e., graph or textual artifacts); kind of relationship allowed between the source and target artifacts; changes allowed by the tool; and execution strategy with respect to several aspects (e.g., automation, availability of an explicit backward transformation). Diskin et al. [178] present a three-dimensional taxonomy of bidirectional model synchronization. Each point in the dimension refers to a specific synchronization semantics with an underlying algebraic model and the respective requirements for the change propagation operations and their properties. Hildebrandt et al. [137] conducted a comparative study of TGG tools, emphasizing the importance of the correctness and the completeness for certain classes of TGGs. Several other surveys review model transformation approaches and tools (e.g., [134,138,141,179]). A number of studies have reviewed a limited subset of available tools and compared them based on a number of features [136,137].

Compared to existing work, our study presents a survey and classification that is more comprehensive and detailed in terms of: (1) a large number of tools and approaches, (2) a wider range of categories and facets.

8 Conclusion

In this paper, we classified and compared 60 model transformation tools with respect to a range of important tool facets grouped into six categories: *general*, *model-level*, *transformation*, *user experience*, *collaboration support*, and *run-time requirements*. We presented a thorough study of model transformation concepts and their classifications, along with detailed descriptions of how existing tools support these concepts. Our analysis and the corresponding website can help potential users to quickly assess which tools they should consider for a given application, whether those tools are still supported, and whether they support all of the facets of interest to them.

The survey allowed us to make a number of observations that may be of interest to the broader modeling community. For example:

- Many of the tools are only maintained for a short period of time after their initial release. This may be because they never reach their intended audience. This study may encourage the community to build on or reuse these tools, especially given the effort required to develop a new tool from scratch. In this regard, our study provides a comprehensive repository that can help researchers and practitioners find the tools that are closest to their needs.
- Some important facets, such as incrementality, built-in traceability, and verification and validation are not supported by a large number of tools, indicating a lack of applicable research results in these areas. These facets thus present an opportunity for future research in model transformation.
- Many of the tools presented lack adequate tutorials or examples in order to fully explain the tool's features. These shortcomings are often a significant barrier to adoption.

To best of our knowledge, this research presents the most comprehensive survey of model transformation and related tools, and provides a complete picture of the current state-of-the-art and practice in model transformation. The complete findings from this study are available on a website[2], which can help interested users find suitable model transformation tools using, e.g., supported facets as search criteria. We plan to keep this website up-to-date as tools evolve.

Acknowledgements The assistance of the following people is gratefully acknowledged: Kevin Lano, Jim Steel, Antonio Cicchetti, Jesús Manuel Almendros Jiménez, Cedric Dumoulin, Alcino Cunha, Nuno Macedo, Li Dan, Sreedhar S. Reddy, Bernhard Schätz, Rusi Popov, Sven Efftinge, Peter Friese, Janne Luoma, Juha-Pekka Tolvanen, Christopher Gerking, Didier Vojtisek, Desfray Philippe, Timothy Lethbridge, Thomas Degueule, Donatas Mazeika, Paul Boocock, Jens von Pilgrim, Hui Song, Joël Cheuoua, Gary Reeves, Arend Rensink, Ed Willink, Hans Vangheluwe, Simon Van Mierlo, Claudia Ermel, Peter Braun, Soichiro Hidaka, Zhenjiang Hu, Andy Schürr, Gergely Varro, Joel Greenyer, Wimmer Manuel, Gehan M.K. Selim, Pieter Van Gorp, Jesús Sánchez Cuadrado, Dimitris Kolovos, Mezei Gergely, William Piers, Albert Zündorf, Edgar Jakumeit, Christian Krause, Jean-Michel Bruel, the Blu Age customer service, Audris Kalnins, Dermot O'Bryan, Thomas Capelle, Cédric Brun, and Reto Carrara. This work is supported in part by the Natural Sciences and Engineering Research Council of Canada.

[2] http://www.mdetools.com.

Appendix A: Summary of tool attributes

Table 9 Summary of tool attributes

Tool	Model-level (Table 4)									General (Table 3)					
	ML	MML	MC	MQ	MME	MR	CS	RE	RT	UP	L	TA	TES	SR	SEC
UML-RSDS	b	a	s	s	s	s	ah	s	s	a	g	a	s	abd	s
Tefkat	s	ac	s	s	s	a	abf	s	s	c	b	a	b	abcde	s
JTL	s	c	s	s	s	a	afh	s	s	a	f	a	s	bd	s
PTL	s	c	s	s	s	a	afh	s	s	b	g	a	s	bd	s
ModTransf	s	a	s	s	s	b	ahk	s	s	c	f	a	s	abd	s
Echo	s	c	s	s	s	a	adh	s	s	b	a	a	s	abd	b
QVTR-XSLT	s	a	s	a	s	s	adh	s	s	c	g	a	s	abd	s
ModelMorf	s	e	s	s	s	c	adh	s	s	c	gh	a	s	abd	s
mediniQVT	s	c	s	a	s	a	adh	s	s	c	ah	n	s	abcde	n
PETE	s	c	s	s	s	a	a	s	s	c	g	a	s	bcd	s
TXL	s	e	s	s	s	s	a	s	s	a	g	a	b	abcde	s
ModelAnt	s	a	ac	a	s	b	agk	a	s	b	a	b	b	abd	s
Xtend	s	c	s	a	s	c	n	s	s	a	a	ab	b	abcde	s
MetaEdit+	abcdefgh	e	abcd	a	a	c	a	a	a	a	hi	ab	a	abcde	ab
QVTo-Eclipse	s	c	s	s	s	a	ch	s	s	a	a	a	s	abce	b
Kermeta2	s	ce	s	a	a	a	abh	s	s	d	a	a	s	abcde	s
Modelio	bdgeh	c	s	s	s	a	ah	a	a	a	fhi	ab	a	abcde	b
Umple	b	ce	ad	s	s	c	ah	a	s	a	e	ab	b	abcde	s
Melange	s	c	s	s	a	a	ah	s	s	a	a	a	b	abd	ab
MagicDraw	abdg	a	s	a	s	ac	ach	a	a	a	hi	ab	a	abcde	b
JAMDA	s	n	s	s	s	s	a	s	s	c	c	a	s	abd	s
SmartQVT	s	c	s	s	s	a	ac	s	s	c	a	a	s	abc	s
SiTra	s	n	s	s	s	s	n	s	s	b	f	a	s	abd	s
Mitra2	s	c	s	s	s	a	ah	s	s	b	a	a	s	d	s
JQVT	s	c	s	s	s	s	f	s	s	c	a	a	s	s	s
Together	abfg	ac	acd	n	s	a	acegh	a	a	a	hi	ab	a	abcde	n
Merlin	s	c	s	s	s	a	af	s	s	c	a	a	s	abd	s
MOFScript	s	c	s	s	s	a	ag	s	s	c	a	a	s	ade	n
GROOVE	s	c	s	s	s	s	a	s	s	b	c	a	b	abd	s
UMLX	s	ac	s	s	s	a	adeh	s	s	b	a	a	b	abcde	s
AToM3	e	e	s	s	a	s	hi	s	s	d	b	a	s	abd	s
AToMPM	b	e	s	a	a	s	hij	s	s	b	b	a	b	abd	s
AGG	h	e	s	s	s	s	ai	s	s	a	a	ab	b	abd	s
BOTL	s	a	s	s	s	s	af	s	s	c	b	a	s	a	s
GRoundTram	s	d	s	a	s	s	h	s	a	b	e	a	b	abd	s
eMoflon	s	c	s	s	s	ac	a	s	s	a	ab	ab	b	abcd	s
MoTE	s	c	s	s	s	a	ah	s	a	a	f	a	b	abd	s
GReAT	s	e	s	n	s	s	n	s	s	b	f	a	s	abcde	s
TGGInterpreter	s	c	s	s	s	a	afh	s	s	b	g	a	s	abd	s
MOMoT	s	c	s	s	s	a	a	s	s	b	a	a	b	bd	s
EMorF	s	c	s	a	s	a	ah	s	s	c	a	n	s	abd	n
MoTMoT	s	ae	s	a	s	b	ahk	s	s	c	b	a	s	abd	s

Table 9 continued

Tool	Model-level (Table 4)									General (Table 3)					
	ML	MML	MC	MQ	MME	MR	CS	RE	RT	UP	L	TA	TES	SR	SEC
DSLTrans	s	c	s	s	s	a	a	s	s	b	a	a	s	ab	s
VIATRA	s	c	s	a	s	a	ah	s	s	b	a	ab	b	abcde	ab
Eclectic	s	c	s	s	s	ac	a	s	s	b	b	a	s	bd	s
Epsilon	s	c	abc	a	s	ab	ahj	s	s	b	a	ab	b	abcde	b
AGE	s	ac	s	s	s	ac	ah	s	s	c	a	ab	s	abd	s
VMTS	df	e	s	a	a	c	ch	s	s	b	g	ab	b	abd	b
ATL	s	acd	s	a	s	ab	afh	s	s	a	a	ab	b	abcde	b
Fujaba	b	a	s	s	s	s	a	a	a	b	e	a	s	abd	s
GrGen.NET	fh	c	s	s	s	c	a	s	s	b	b	ab	b	abcde	a
Henshin	s	ce	s	s	s	a	ahi	s	s	a	a	ab	b	abcd	s
Blu Age	bf	ce	ac	a	s	a	afghij	a	a	a	h	b	a	abcd	ab
MOLA	s	e	s	s	s	ac	ah	s	s	b	g	a	b	abd	s
SPARX	bdgh	c	ac	s	s	a	ah	a	a	a	ghi	ab	a	abcde	a
MDWorkbench	s	c	s	s	s	a	ah	a	s	a	ghi	ab	a	abd	a
Acceleo	s	c	s	a	s	ab	aghk	a	s	a	a	ab	b	abcde	s
AndroMDA	s	a	s	s	s	n	ak	s	s	c	d	n	s	abde	s
Xpand	s	c	s	s	s	n	a	s	s	d	a	n	s	abcde	s
Actifsource	b	c	s	s	s	a	a	s	s	a	gh	ab	a	abcde	b

Tool	Transformation (Table 5)															
	MTLX	O	C	RS	RO	RAC	T	D	V	VA	TR	IU	CT	LT	TS	CH
UML-RSDS	ab	ab	abcd	abi	ab	a	ab	bc	abcdeg	s	b	a	s	s	s	abcd
Tefkat	b	b	abcd	bdi	ac	b	a	c	s	a	ab	s	s	s	s	abcd
JTL	ab	ab	abcd	bcd	a	b	ab	bc	ac	s	ab	a	s	s	s	abcd
PTL	b	b	abcd	bcf	a	a	ab	c	af	a	a	s	s	s	s	abcd
ModTransf	b	ad	ad	abcfgi	ab	a	ab	c	s	s	s	s	s	s	s	abcd
Echo	b	b	a	d	s	b	a	bc	acfg	b	s	a	s	s	s	abcd
QVTR-XSLT	a	ab	ac	bf	bc	a	ab	c	s	s	a	s	s	s	s	abcd
ModelMorf	b	ab	ad	abfhi	abc	a	ab	abc	a	s	a	a	s	s	s	abcd
mediniQVT	b	n	n	bc	n	n	n	bc	ag	n	a	a	s	s	s	abcd
PETE	b	b	a	bcdfi	a	b	a	c	n	s	b	s	s	s	s	abcd
TXL	b	abcde	abcd	abcfh	c	a	ab	c	s	s	s	s	s	s	s	abcd
ModelAnt	n	bcde	ab	ac	n	a	ab	c	s	s	s	s	s	s	s	abcd
Xtend	b	abcde	abcd	acefg	abc	a	ab	c	s	ab	b	a	a	s	s	abcd
MetaEdit+	a	bcd	abcd	acdfh	ab	a	ab	c	a	b	a	a	a	s	s	abcd
QVTo-Eclipse	b	ab	abcd	aci	abc	a	ab	c	s	s	a	n	a	s	s	abcd
Kermeta2	b	abc	abcd	afg	ab	abcd	ab	c	a	a	b	s	s	s	s	abcd
Modelio	b	abcde	abcd	s	s	n	ab	c	a	a	ab	a	s	s	s	abcd
Umple	ab	abcde	abd	a	b	a	ab	abc	acf	ab	a	a	a	s	s	abcd
Melange	b	abc	abcd	acfg	a	a	ab	c	ag	s	b	s	s	s	s	abcd
MagicDraw	a	abcde	abcd	ac	a	a	ab	c	acd	ab	a	a	a	s	s	abcd
JAMDA	n	bd	ad	ac	ab	a	a	c	s	s	s	s	s	s	s	abcd
SmartQVT	b	n	n	afi	ab	n	n	c	s	s	a	s	s	s	s	abcd
SiTra	n	b	n	abcf	n	ad	n	c	s	s	a	s	s	s	s	abcd
Mitra2	b	ab	abcd	abceg	a	ad	ab	c	s	s	a	s	s	s	s	abcd

Table 9 continued

Tool	Transformation (Table 5)															
	MTLX	O	C	RS	RO	RAC	T	D	V	VA	TR	IU	CT	LT	TS	CH
JQVT	n	b	ab	acf	b	a	a	c	s	s	s	s	s	s	s	abcd
Together	b	abcde	abcd	n	n	n	ab	c	acfg	ab	a	a	a	a	s	abcd
Merlin	ab	abd	abd	acf	s	a	a	c	s	s	a	a	s	s	s	abcd
MOFScript	b	abcde	n	acg	n	a	n	c	n	s	s	s	s	s	s	abcd
GROOVE	ab	a	a	abcdefghi	ac	abcd	b	c	abcef	b	b	s	s	s	s	abcd
UMLX	a	b	abcd	abcdfgh	bc	b	ab	c	s	ab	a	a	s	s	s	abcd
AToM3	a	n	a	bdeh	s	bd	n	c	n	s	b	n	s	s	s	abcd
AToMPM	a	a	a	adeg	s	bd	ab	c	s	b	b	a	a	s	s	abcd
AGG	a	ab	abcd	bdeh	bc	cd	b	c	abcef	b	b	s	a	s	a	abcd
BOTL	a	b	a	bdh	s	b	b	bc	abcde	b	s	s	a	s	s	abcd
GRoundTram	ab	ab	a	acf	c	s	b	bc	abe	s	a	s	s	s	s	abcd
eMoflon	ab	abc	abcd	acdegh	ab	bcd	ab	bc	ac	s	a	a	s	s	s	abcd
MoTE	a	b	a	bdh	c	a	ab	bc	abcdeg	a	ab	a	a	s	s	abcd
GReAT	a	n	n	acdfh	ac	ab	n	c	n	n	b	n	s	s	s	abcd
TGGInterpreter	a	b	abcd	bdg	ab	c	a	abc	ac	s	b	a	s	s	s	abcd
MOMoT	ab	ac	a	bd	ac	c	ab	c	s	s	s	s	s	s	a	abcd
EMorF	a	n	a	n	n	n	ab	bc	s	a	a	a	s	s	s	abcd
MoTMoT	a	n	abcd	acg	b	a	ab	c	n	n	b	s	s	s	s	abcd
DSLTrans	ab	b	a	adh	c	n	a	c	abe	s	a	s	s	s	s	abcd
VIATRA	b	abcde	abcd	abcdfgh	ac	bcd	ab	c	abcdfg	a	ab	a	s	a	a	abcd
Eclectic	b	b	abcd	abc	abc	a	ab	c	s	s	b	s	a	s	s	abcd
Epsilon	b	abcde	abcd	abfgh	ab	d	ab	c	a	a	a	a	s	s	s	abcd
AGE	b	bd	abcd	bci	ab	a	ab	c	s	s	b	s	s	s	s	abcd
VMTS	ab	abcd	b	acefgh	ab	ad	ab	c	ab	b	b	a	s	s	s	abcd
ATL	b	ab	abcd	abdfi	ab	a	ab	c	b	s	b	s	s	s	s	abcd
Fujaba	a	abd	a	acfg	abc	abc	ab	c	ac	ab	b	a	s	s	s	abcd
GrGen.NET	b	acd	a	adefghi	ac	abcd	ab	c	s	ab	b	s	a	s	s	abcd
Henshin	a	abd	abcd	acdgh	a	bd	ab	c	abc	a	b	s	a	s	s	abcd
Blu Age	ab	abcde	abcd	abcdefghi	abc	abcd	ab	bc	acf	a	a	a	a	a	a	abcd
MOLA	a	ab	a	acg	ac	ab	ab	c	s	s	b	s	s	s	s	abcd
SPARX	a	abde	abd	n	n	n	ab	c	s	ab	a	a	s	s	s	abcd
MDWorkbench	b	abce	abcd	acg	b	a	ab	c	s	a	a	s	a	s	s	abcd
Acceleo	b	cd	ab	acg	ab	bc	a	c	s	s	a	a	s	s	s	abcd
AndroMDA	b	cde	n	n	n	ad	a	c	s	s	s	n	s	s	s	abcd
Xpand	b	cd	a	abefgi	abc	a	a	c	s	s	b	a	n	s	s	abcd
Actifsource	ab	cd	abcd	ac	s	s	a	c	a	a	b	a	s	s	s	abcd

Tool	User Experience (Table 6)							Collaboration Sup.(Table 7)				Runtime Req.(Table 8)			
	UIN	WPM	SYE	SE	LA	AR	PS	TSU	RUT	IN	E	OS	EM	EXM	ED
UML-RSDS	ab	s	s	ab	b	s	c	a	a	b	b	ab	cd	b	n
Tefkat	a	a	ad	bdf	b	s	bd	s	abc	b	s	abc	ad	a	b
JTL	a	a	s	bd	b	s	d	s	s	b	a	abc	a	a	b
PTL	a	a	s	bd	b	s	d	s	a	s	s	abc	ad	a	b
ModTransf	b	s	s	s	a	s	ab	s	abc	b	a	abc	ad	a	b
Echo	ab	a	abcde	bd	b	s	c	s	s	b	s	abc	ad	a	b
QVTR-XSLT	a	a	s	b	b	s	c	s	a	b	s	ab	b	b	b

Table 9 continued

Tool	User Experience (Table 6)							Collaboration Sup.(Table 7)				Runtime Req.(Table 8)			
	UIN	WPM	SYE	SE	LA	AR	PS	TSU	RUT	IN	E	OS	EM	EXM	ED
ModelMorf	b	s	s	b	b	s	c	s	a	b	s	ab	cd	a	b
mediniQVT	ab	a	abcd	abcd	b	n	c	n	n	b	a	abc	ad	a	n
PETE	a	a	s	bd	b	s	d	s	ac	b	s	abc	a	a	b
TXL	b	s	abcd	bdg	b	s	a	s	ac	s	s	abc	ad	ab	a
ModelAnt	s	s	s	s	b	a	b	s	d	b	a	abc	cd	b	b
Xtend	ab	a	abcde	abcdefg	b	s	a	ab	abd	ab	a	abc	abd	b	b
MetaEdit+	ab	a	abde	abd	c	a	a	ab	abcde	ab	a	abc	abd	b	a
QVTo-Eclipse	a	a	acde	bdeg	b	s	a	b	ab	b	a	abc	a	a	b
Kermeta2	a	a	abcde	abdf	b	a	a	ab	abcde	ab	a	abc	ad	b	b
Modelio	a	a	abcde	abcdefg	c	a	ab	ab	s	ab	a	abc	abd	b	a
Umple	ab	a	ae	bd	b	a	a	ab	a	ab	a	abc	ad	b	a
Melange	a	a	abcde	abcdef	b	s	a	s	acd	b	n	abc	a	b	b
MagicDraw	a	a	abcde	abd	c	a	a	ab	ad	ab	a	abc	abd	b	a
JAMDA	s	s	s	s	a	s	a	s	s	s	a	abc	cd	b	b
SmartQVT	a	a	a	b	b	s	a	s	n	b	a	abc	a	b	b
SiTra	s	s	s	s	b	s	a	s	s	s	n	abc	ab	n	b
Mitra2	a	a	acde	bcde	b	s	a	s	a	b	s	abc	a	a	b
JQVT	a	a	ac	bc	b	s	a	s	s	b	s	abc	a	b	b
Together	ab	a	abcde	abcdefg	b	a	a	ab	abcde	ab	a	ab	a	b	b
Merlin	a	a	acd	bcd	b	s	a	s	s	b	a	abc	a	b	b
MOFScript	a	a	ac	bcd	b	s	ab	n	n	b	n	abc	a	b	b
GROOVE	ab	a	ac	bd	b	s	c	s	s	b	a	abc	cd	a	a
UMLX	a	a	s	bdf	b	s	c	s	a	ab	s	abc	a	ab	b
AToM3	a	a	s	n	b	s	c	s	n	n	s	abc	cd	b	b
AToMPM	a	a	s	bd	b	s	b	ab	ab	a	a	abc	cd	ab	b
AGG	ab	a	s	abd	b	s	c	s	a	ab	a	abc	cd	a	a
BOTL	a	s	s	s	b	s	c	s	b	b	s	abc	cd	b	b
GRoundTram	ab	a	a	bd	b	s	ac	s	s	b	n	abc	cd	a	a
eMoflon	ab	a	ac	bcd	c	s	c	a	s	b	a	a	a	ab	b
MoTE	a	a	s	b	b	s	c	s	s	b	s	abc	ad	ab	b
GReAT	a	a	n	n	b	s	c	s	n	n	a	a	b	n	b
TGGInterpreter	a	a	s	bd	b	s	c	s	s	b	a	abc	a	a	b
MOMoT	a	s	abcde	abce	b	s	ab	s	ab	ab	s	abc	a	ab	b
EMorF	a	a	n	bcd	b	s	c	n	n	n	a	abc	a	a	b
MoTMoT	ab	a	abce	e	b	n	a	s	abce	b	a	abc	bd	b	b
DSLTrans	a	a	s	s	b	s	c	s	ae	s	n	abc	a	b	b
VIATRA	ab	a	abcde	bcdfg	c	s	c	s	abcde	ab	a	abc	a	ab	b
Eclectic	a	a	abcd	bef	b	s	a	ab	abc	b	a	abc	a	b	b
Epsilon	a	a	acde	bcdfg	b	s	ab	s	abce	ab	a	abc	ad	a	b
AGE	a	a	abce	bf	b	s	a	ab	ab	b	a	abc	ad	a	b
VMTS	ab	a	abc	bd	b	a	abc	b	ac	b	a	a	bd	ab	b
ATL	a	a	abcde	bcd	b	s	a	s	n	b	a	abc	a	b	b
Fujaba	a	a	abcd	bcdefg	b	s	ac	ab	abcde	ab	a	abc	ad	b	a
GrGen.NET	b	s	a	bdg	b	s	ac	s	abcd	ab	a	abc	cd	ab	b
Henshin	a	a	s	abd	b	s	c	s	ae	b	a	abc	a	ab	b
Blu Age	ab	a	abcde	abcdefg	c	a	abc	ab	abcde	ab	a	a	a	b	b

Table 9 continued

Tool	User Experience (Table 6)							Collaboration Sup.(Table 7)				Runtime Req.(Table 8)			
	UIN	WPM	SYE	SE	LA	AR	PS	TSU	RUT	IN	E	OS	EM	EXM	ED
MOLA	a	a	c	abd	b	s	ac	s	ae	b	s	a	a	b	b
SPARX	ab	a	abcde	bcde	b	a	ab	ab	n	ab	a	abc	ab	b	a
MDWorkbench	a	a	abcde	bd	b	a	a	ab	n	ab	a	abc	a	b	b
Acceleo	a	a	abcde	abcdeg	b	a	ab	s	a	b	a	abc	abd	b	b
AndroMDA	b	s	s	s	b	s	a	s	n	s	a	abc	bd	n	b
Xpand	a	a	abcde	n	b	s	a	n	n	b	a	abc	a	b	b
Actifsource	a	a	abcde	abcdefg	b	a	a	ab	n	ab	a	abc	a	b	b

s (No support)
n (Information not available)

References

1. Kahani, N., Cordy, J.: Comparison and evaluation of model transformation tools. In: Technical Report 2015-627, Queen's University, pp. 1–42 (2015)
2. Bagherzadeh, M., Hili, N., Dingel, J.: Model-level, platform-independent debugging in the context of the model-driven development of real-time systems. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 419–430 (2017)
3. Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G., Syriani, E., Wimmer, M.: Model transformation intents and their properties. In: Software and Systems Modeling, pp. 1–38 (2014)
4. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3), 621–645 (2006)
5. Mens, T., Gorp, P.V.: A taxonomy of model transformation. Electron. Notes Theor. Comput. Sci. **152**, 125–142 (2006)
6. Salem, R.B., Grangel, R., Bourey, J.: A comparison of model transformation tools: Application for transforming GRAI extended Actigrams into UML activity diagrams. In: Computers in Industry, pp. 682–693 (2008)
7. Macedo, N., Jorge, T., Cunha, A.: A feature-based classification of model repair approaches. IEEE Trans. Softw. Eng. **43**(7), 615–640 (2017)
8. Jakumeit, E., Buchwald, S., Wagelaar, D., Dan, L., Hegedüs, Á., Herrmannsdörfer, M., Horn, T., Kalnina, E., Krause, C., Lano, K., Lepper, M., Rensink, A., Rose, L., Wätzoldt, S., Mazanek, S.: A survey and comparison of transformation tools based on the transformation tool contest. Sci. Comput. Program. **85**, 41–99 (2014)
9. Taentzer, G., Ehrig, K., Guerra, E., Lara, J., Lengyel, L., Levendovszky, T., Prange, U., Varro, D., Varró-Gyapay, S.: Model transformation by graph transformation: A comparative study. In: Proceedings Workshop Model Transformation in Practice, Montego Bay, Jamaica, pp. 1–48 (2005)
10. Hidaka, S., Tisi, M., Cabot, J., Hu, Z.: Feature-based classification of bidirectional transformation approaches. Softw. Syst. Model. **15**(3), 1–22 (2015)
11. Gomes, C., Barroca, B., Amaral, V.: Classification of model transformation tools: pattern matching techniques, pp. 619–635 (2014)
12. Biehl, M.: Literature study on model transformations. In: Royal Institute of Technology, pp. 1–24 (2010)
13. Mesa, J. M. V.: M2DAT: A technical solution for model-driven development of web information systems.Ph.D. thesis, University of Rey Juan Carlos (2009)
14. Uhl, A.: Model-driven development in the enterprise. IEEE Softw. **1**, 46–49 (2008)
15. Huber, P.: The model transformation language jungle: An evaluation and extension of existing approaches. In: Master thesis, University of Vienna (2008)
16. Rothenberg, J., Widman, L., Loparo, K., Nielsen, N.: The nature of modeling, pp. 1–18 (1989)
17. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice. Synth. Lect. Softw. Eng. **1**(1), 1–182 (2012)
18. Unified Modeling Language (UML) http://www.uml.org. Accessed 16 Feb 2018
19. Aers, K.: Graphiti and GMF compared: revisiting the graph editor. In: EclipseCon 2011, Santa Clara, California (2011)
20. Viyović, V., Maksimović, M., Perisić, B.: Sirius: A rapid development of DSM graphical editor. In: IEEE 18th International Conference on Intelligent Engineering Systems, pp. 233–238 (2014)
21. Efftinge, S., Völter, M.: oAW xText: A framework for textual DSLs. In: Workshop on Modeling Symposium at Eclipse Summit, pp. 118–121 (2006)
22. Henriksson, J., Johannes, J., Zschaler, S., Asmann, U.: Reuseware-adding modularity to your language of choice. J. Object Technol. **6**(9), 127–146 (2007)
23. Kleppe, A.: A language description is more than a metamodel. In: Fourth International Workshop on Software Language Engineering, pp. 1–9 (2007)
24. ArcStyler: The leading platform for model driven architecture (MDA) http://www.omg.org/mda/mda_files/ArcStyler5_Whitepaper_220205.pdf. Accessed 16 Feb 2018
25. Patrascoiu, O.: YATL: yet another transformation language-reference manual version 1.0. In: Technical Report No. 2-04 (2004)
26. Codagen Architect http://www.omg.org/mda/mda_files/codagen_technologies.htm. Accessed 20 Feb 2018
27. OptimalJ http://www.compuware.com. Accessed 16 Feb 2018
28. Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E.: Graphical definition of in-place transformations in the eclipse modeling framework. In: Proceeding of the International Conference on Model Driven Engineering Languages and Systems (MoDELS06) 425439 (2006)
29. FUUT-je http://www.eclipse.org/gmt/. Accessed 01 May 2017
30. Boronat, A.: MOMENT: a formal framework for MOdel managEMENT, In: Ph.D. thesis in Computer Science, University of Politécnica de Valéncia, pp. 1–287 (2007)
31. Sánchez-Barbudo, A., Sánchez, E., Roldán, V., Estévez, A., Roda, J.: Providing an open virtual-machine-based QVT implemen-

tation. In: Proceedings of the V Workshop on Model-Driven Software Development, pp. 42–51 (2008)
32. b+m ArchitectureWare http://www.omg.org/mda/mda_files/b+m_OMGCommittment.pdf. Accessed 16 Feb 2018
33. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: the missing link of MDA. In: Graph Transformation, pp. 90–105 (2002)
34. QVTd https://projects.eclipse.org/projects/modeling.mmt.qvtd. Accessed 16 Feb 2018
35. Vlad, A., Störrle, H., Strüber, D.: VMTL: A language for end-user model transformation. In: Software and Systems Modeling, pp. 1–29 (2016)
36. Acerbis, R., Bongio, A., Brambilla, M., Butti, S.: Webratio 5: An Eclipse-based case tool for engineering web applications, pp. 501–505. In. In Web, Engineering (2007)
37. UMT http://umt-qvt.sourceforge.net/. Accessed 16 Feb 2018
38. Roy, C., Cordy, J., Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: a qualitative approach. Sci. Comput. Program. **74**(7), 470–495 (2009)
39. Kastenberg, H., Rensink, A.: Model checking dynamic states in GROOVE. In: International SPIN Workshop on Model Checking of Software, McGill University, pp. 299–305 (2006)
40. Bruneliere, H., Cabot, J., Jouault, F., Madiot, F.: MoDisco: A generic and extensible framework for model driven reverse engineering. In: Proceedings of the IEEE/ACM International Conference on Automated software engineering, pp. 173–174 (2010)
41. Lano, K., Kolahdouz-Rahimi, S.: Specification and verification of model transformations using UML-RSDS, pp. 199–214 (2010)
42. Lawley, M., Steel, J.: Practical declarative model transformation with Tefkat, pp. 139–150 (2006)
43. Romina, E., Pierantonio, A., Rosa, G.: Managing uncertainty in bidirectional model transformations, in: Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering, pp. 49–58 (2015)
44. Almendros-Jiménez, J. M., Iribarne, L., López-Fernández, J., Mora-Segura, A.: PTL: A model transformation language based on logic programming. In: Journal of Logical and Algebraic Methods in Programming, pp. 89–105 (2015)
45. Bonde, L., Dumoulin, C., Dekeyser, J.: Metamodels and MDA transformations for embedded systems. In: Advances in Design and Specification Languages for SoCs, pp. 89–105 (2005)
46. Macedo, N., Cunha, A.: Implementing QVT-R bidirectional model transformations using Alloy. In: Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering, pp. 297–311 (2013)
47. Li, D., Li, X., Stolz, V.: QVT-based model transformation using XSLT. In: SIGSOFT Software Engineering Notes, pp. 1–8 (2011)
48. Reddy, S., Venkatesh, R., Ansari, Z.: A relational approach to model transformation using QVT relations. In: TATA Research Development and Design Centre, pp. 1–15 (2006)
49. medini QVT http://projects.ikv.de/qvt/wiki. Accessed 16 Feb 2018
50. Schätz, B.: Formalization and rule-based transformation of EMF Ecore-based models, pp. 227–244 (2009)
51. Paige, R., Radjenovic, A.: Towards model transformation with TXL. In: Metamodelling for MDA, pp. 162–177 (2003)
52. ModelAnt http://mdatools.net/blog/modelant. Accessed 16 Feb 2018
53. Xtend https://eclipse.org/xtend/index.html
54. Kelly, S., Lyytinen, K., Rossi, M.: Metaedit+ a fully configurable multi-user and multi-tool case and came environment. Adv. Inf. Syst. Eng. **1080**, 1–21 (1996)
55. Gerking, C., Heinzemann, C.: Solving the movie database case with QVTo. In: TTC, pp. 98–102 (2014)
56. Drey, Z., Faucher, C., Fleurey, F., Mahé, V., Vojtisek, D.: Kermeta language reference manual, pp. 1–84 (2010)
57. Modelio http://www.modeliosoft.com. Accessed 16 Feb 2018
58. Forward, A., Lethbridge, T., Brestovansky, D.: Improving program comprehension by enhancing program constructs: An analysis of the Umple language. In: ICPC, pp. 311–312 (2009)
59. Degueule, T., Combemale, B., Blouin, A., Barais, O., Jézéquel, J.: Melange: A meta-language for modular and reusable development of DSLs. In: 8th International Conference on Software Language Engineering (SLE), pp. 65–75 (2015)
60. MagicDraw http://www.nomagic.com. Accessed 16 Feb 2018
61. Jamda http://jamda.sourceforge.net/#documentation
62. SmartQVT https://sourceforge.net/projects/smartqvt. Accessed 16 Feb 2018
63. SiTra http://www.cs.bham.ac.uk/~bxb/Sitra/index.html. Accessed 16 Feb 2018
64. Pilgrim, J.V.: Computerunterstützte Modelltransformationen. Ph.D. thesis in Computer Science, Fernuniversität Hagen (2010)
65. JQVT https://sourceforge.net/projects/jqvt/. Accessed 16 Feb 2018
66. Merlin https://sourceforge.net/projects/merlingenerator/?source=navbar. Accessed 16 Feb 2018
67. Together http://www.borland.com/Products/Requirements-Management/Together. Accessed 16 Feb 2018
68. MOFScript https://marketplace.eclipse.org/content/mofscript-model-transformation-tool. Accessed 16 Feb 2018
69. Rensink, A.: The GROOVE simulator: a tool for state space generation. In: Applications of Graph Transformations with Industrial Relevance, pp. 479–485 (2004)
70. Willink, E. D.: UMLX: A graphical transformation language for MDA. In: Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications, pp. 13–24 (2003)
71. Lara, J., Vangheluwe, H.: AToM3: A tool for multi-formalism and meta-modelling. In: FASE, pp. 174–188 (2002)
72. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, S. V., Ergin, H.: AToMPM: A web-based modeling environment. In: Demos/Posters/Student Research MoDELS, pp. 21–25 (2013)
73. Ermel, C., Rudolf, M., Taentzer, G.: The AGG approach: Language and environment. Appl. Lang. Tools. **2**, 551–603 (1999)
74. Braun, P., Marschall, F.: Transforming object oriented models with BOTL. In: Electronic Notes in Theoretical Computer Science, pp. 103–117 (2003)
75. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Nakano, K.: GRoundTram: An integrated framework for developing well-behaved bidirectional model transformations. In: 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 480–483 (2011)
76. Lauder, M., Anjorin, A., Varró, G., Schürr, A.: Bidirectional model transformation with precedence triple graph grammars, pp. 287–302 (2012)
77. Giese, H., Hildebrandt, S., Lambers, L.: Bridging the gap between formal semantics and implementation of triple graph grammars. Softw. Syst. Model. **13**(1), 273–299 (2014)
78. GReAT http://www.isis.vanderbilt.edu/tools/great. Accessed 16 Feb 2018
79. Greenyer, J., Kindler, E.: Reconciling TGGs with QVT. In: Model Driven Engineering Languages and Systems, pp. 16–30 (2007)
80. Fleck, M., Troya, J., Wimmer, M.: Marrying search-based optimization and model transformation technology. In: Proceedings of the First North American Search Based Software Engineering Symposium, pp. 1–16 (2015)
81. Klassen, L., Wagner, R.: EMorF-A tool for model transformations. In: Electronic Communications of the EASST, pp. 1–6 (2012)
82. Barroca, B., Lúcio, L., Amaral, V., Félix, R., Sou, V.: Dsltrans: A Turing incomplete transformation language. In: Software Language Engineering, 29630 (2011)

83. Gorp, G. V.: Model-driven development of model transformations. Ph.D. thesis, University of Antwerp (2008)
84. Varró, D., Balogh, A.: The model transformation language of the VIATRA2 framework. Sci. Comput. Program. **68**(3), 214–234 (2007)
85. Cuadrado, J.: Towards a family of model transformation languages, pp. 176–191 (2012)
86. Kolovos, D., Paige, R., Polack, F.: The Epsilon transformation language. In: Theory and Practice of Model Transformations, pp. 46–60 (2008)
87. Cuadrado, J., Molina, J., Tortosa, M.: Rubytl: A practical, extensible transformation language. In: Model Driven Architecture-Foundations and Application, pp. 158–172 (2006)
88. Levendovszky, T., Lengyel, L., Mezei, G., Charaf, H.: A systematic approach to metamodeling environments and model transformation systems in VMTS. In: Electronic Notes in Theoretical Computer Science, pp. 65–75 (2005)
89. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. In: Science of Computer Programming, pp. 31–39 (2008)
90. Nickel, U., Niere, J., Zündorf, A.: The FUJABA environment. In: Proceedings of the 22nd International Conference on Software Engineering, pp. 742–745 (2000)
91. Jakumeit, E., Buchwald, S., Kroll, M.: Grgen.net: The expressive, convenient and fast graph rewrite system. In: International Journal on Software Tools for Technology Transfer, pp. 263–271 (2010)
92. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: Advanced concepts and tools for in-place EMF model transformations. In: Model Driven Engineering Languages and Systems, pp. 121–135 (2010)
93. Blu Age http://www.bluage.com/en/en_home.html. Accessed 16 Feb 2018
94. Kalnins, A., Barzdins, J., Celms, E.: Model transformation language MOLA. In: Model Driven Architecture, pp. 62–76 (2005)
95. Enterprise Architect http://www.sparxsystems.com. Accessed 16 Feb 2018
96. MDWorkbench http://sodius.com/products-overview/mdworkbench. Accessed 16 Feb 2018
97. Brun, C., Pierantonio, A.: Model differences in the Eclipse modeling framework. In: The European Journal for the Informatics Professional, pp. 29–34 (2008)
98. AndroMDA http://andromda.sourceforge.net. Accessed 16 Feb 2018
99. Xpand https://eclipse.org/modeling/m2t/?project=xpand. Accessed 16 Feb 2018
100. Varró, D., Hegedüs, G.B.A., Horváth, A., Ráth, I., Ujhelyi, Z.: Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. Softw. Syst. Model. **15**(9), 609–629 (2016)
101. Actifsource, http://www.actifsource.com. Accessed 16 Feb 2018
102. Query/views/transformation language (QVT) http://www.omg.org/spec/QVT. Accessed 16 Feb 2018
103. Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H.J., Kuske, S., Plump, D., Schürr, A., Taentzer, G.: Graph transformation for specification and programming. Sci. Comput. Program. **31**(1), 1–54 (1999)
104. Schürr, A.: Specification of graph translators with triple graph grammars. In: Graph-Theoretic Concepts in Computer Science, pp. 151–163 (1995)
105. Roser, S., Lautenbacher, F., Bauer, B.: Generation of workflow code from DSMs. In: Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling, pp. 1–11 (2007)
106. Pearson, H.: Open source licences: open source–the death of proprietary systems? Comput. Law Secur. Rev. **16**, 151–156 (2000)
107. Eramo, R., Marinelli, R., Pierantonio, A.: Towards a taxonomy for bidirectional transformation. In: SATToSE, pp. 122–131 (2014)
108. Kahani, N., Bagherzadeh, M., Dingel, J., Cordy, J.: The problems with Eclipse modeling tools: a topic analysis of eclipse forums. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 227–237 (2016)
109. Ahmad, M., Bruel, J., Laleau, R., Gnaho, C.: Using RELAX SysML and KAOS for ambient systems requirements modeling. In: Procedia Computer Science, pp. 474–481 (2012)
110. xtUML https://xtuml.org. Accessed 16 Feb 2018
111. Kahani, N., Hili, N., Cordy, J., Dingel, J.: Evaluation of UML-RT and Papyrus-RT for modelling self-adaptive systems. In: Proceedings of the 9th International Workshop on Modelling in Software Engineering, pp. 12–18 (2017)
112. Peterson, J.: Petri Net theory and the modeling of systems. In: Prentice Hall PTR (1981)
113. Business Process Model and Notation (BPMN) http://www.bpmn.org. Accessed 16 Feb 2018
114. Meta-Object Facility (MOF) http://www.omg.org/mof. Accessed 16 Feb 2018
115. Eclipse Modeling Framework (EMF) https://eclipse.org/modeling/emf. Accessed 16 Feb 2018
116. Kernel Meta-Meta Model (KM3) https://wiki.eclipse.org/KM3. Accessed 16 Feb 2018
117. Stephan, M., Cordy, J.: A survey of model comparison approaches and applications. In: Modelsward, pp. 265–277 (2013)
118. Bergmann, G.: Translating ocl to graph patterns, pp. 670–686 (2014)
119. Cetinkaya, D., Verbraeck, A.: Metamodeling and model transformations in modeling and simulation. In: Proceedings of the Winter Simulation Conference, pp. 3048–3058 (2011)
120. CDO http://eclipse.org/cdo. Accessed 16 Feb 2018
121. Blanc, X., Gervais, M., Sriplakich, P.: Model bus: Towards the interoperability of modelling tools. In: Model Driven Architecture, pp. 17–32 (2005)
122. EMFStore http://eclipse.org/emfstore. Accessed 16 Feb 2018
123. Benelallam, A., Gómez, A., Sunyé, G., Tisi, M., Launay, D.: Neo4EMF, a scalable persistence layer for EMF models. In: European Conference on Modelling Foundations and Applications, pp. 230–241 (2014)
124. NetBeans Meta-data Repository (MDR) https://netbeans.org. Accessed 16 Feb 2018
125. Canonical XMI http://www.omg.org/spec/XMI/2.5.1. Accessed 16 Feb 2018
126. Human Usable Textual Notation (HUTN) http://www.omg.org/spec/HUTN. Accessed 16 Feb 2018
127. Java Meta-data Interface (JMI) http://download.oracle.com/otn-pub/jcp/7791-jmi-1.0-prd-spec-oth-JSpec/jmi-1_0-prd-spec-update.pdf. Accessed 16 Feb 2018
128. Diagram Definition Specification (DD) http://www.omg.org/spec/DD. Accessed 16 Feb 2018
129. MOF model to text transformation language http://www.omg.org/spec/MOFM2T. Accessed 16 Feb 2018
130. Common Warehouse Meta-model (CWM) http://www.omg.org/spec/CWM. Accessed 16 Feb 2018
131. Object Constraint Language, http://www.omg.org/spec/OCL. Accessed 16 Feb 2018
132. Sendall, S., Küster, J.: Taming model round-trip engineering. In: Proceedings of Workshop on Best Practices for Model-Driven Software Development, pp. 1–13 (2004)
133. Hettel, T., Lawley, M., Raymond, K.: Model synchronisation: Definitions for round-trip engineering, in: International Conference on Theory and Practice of Model Transformations, pp. 31–45 (2008)
134. Syriani, E.: A multi-paradigm foundation for model transformation language engineering. Ph.D. thesis in Computer Science, McGill University, pp. 1–291 (2011)

135. Cuadrado, J. S., Molina, J. G.: A phasing mechanism for model transformation languages. In: Proceedings of the 2007 ACM Symposium on Applied Computing, SAC '07 (2007)
136. Jilani, A., Usman, M., Halim, Z.: Model transformations in model driven architecture. In: Universal Journal of Computer Science and Engineering Technology, pp. 50–54 (2010)
137. Hildebrandt, S., Lambers, L., Giese, H., Rieke, J., Greenyer, J., Schäfer, W., Lauder, M., Anjorin, A., Schürr, A.: A survey of triple graph grammar tools. In: International Workshop on Bidirectional Transformations (Bx), pp. 1–17 (2013)
138. Stevens, P.: A landscape of bidirectional model transformations. Gener. Transform. Tech. Softw. Eng. **II**, 408–424 (2008)
139. Macedo, N., Cunha, A., Pacheco, H.: Towards a framework for multidirectional model transformations. In: EDBT/ICDT Workshops, pp. 71–74 (2014)
140. Czarnecki, K., Foster, J.N., Hu, Z., Lämmel, Schürr, A., Terwilliger, J.F.: Bidirectional transformations: a cross-discipline perspective, pp. 260–283 (2009)
141. Leblebici, E., Anjorin, A., Schürr, A., Hildebrandt, S., Rieke, J., Greenyer, J.: A comparison of incremental triple graph grammar tools. In: Electronic Communications of the EASST, pp. 1–15 (2014)
142. Amrani, M., Combemale, B., Lúcio, L., Selim, G.M.K., Dingel, J., Traon, Y.L., Vangheluwe, H., Cordy, J.R.: Formal verification techniques for model transformations: a tridimensional classification. J. Object Technol. **14**(3), 921–928 (2015)
143. Varró, D., Pataricza, A.: Automated formal verification of model transformations. In: CSDUML, pp. 63–78 (2003)
144. Asztalos, M., Lengyel, L., Levendovszky, T.: Towards automated, formal verification of model transformations. In: Proceedings of the Third International Conference on Software Testing, Verification and Validation, ICST '10, pp. 15–24 (2010)
145. Lano, K., Kolahdouz-Rahimi, S., Poernomo, I.: Comparative evaluation of model transformation specification approaches. Int. J. Softw. Inform. **6**(2), 233–269 (2012)
146. Hooper, P.K.: The undecidability of the Turing machine immortality problem. J. Symbol. Logic **31**(2), 219–234 (1966)
147. Assmann, U.: Graph rewrite systems for program optimization. ACM Trans. Program. Lang. Syst. (TOPLAS) **22**(4), 583–637 (2000)
148. Varró, D., Varró-Gyapay, S., Ehrig, H., Prange, U., Taentzer, G.: Termination analysis of model transformations by Petri Nets. In: Graph Transformations, pp. 260–274 (2006)
149. Ehrig, H., Ehrig, K., Lara, J., Taentzer, G., Varró, D., Varró-Gyapay, S.: Termination criteria for model transformation. In: International Conference on Fundamental Approaches to Software Engineering, pp. 49–63 (2005)
150. Jouault, F., Kurtev, I.: Transforming models with ATL. In: International Conference on Model Driven Engineering Languages and Systems, Springer, pp. 128–138 (2005)
151. Rahim, L., Whittle, J.: A survey of approaches for verifying model transformations. Softw. Syst. Model. **14**(2), 1003–1028 (2015)
152. Rensink, A., Schmidt, Á., Varró, D.: Model checking graph transformations: a comparison of two approaches. In: ICGT, pp. 226–241 (2004)
153. Kastenberg, H., Rensink, A.: Model checking dynamic states in GROOVE. In: Model Checking Software, pp. 299–305 (2006)
154. Fleurey, F., Steel, J., Baudry, B.: Validation in model-driven engineering: Testing model transformations. In: First International Workshop on Model, Design and Validation, pp. 29–40 (2004)
155. Auziņš, A., Bārzdiņš, J., Bičevskis, J., Čerāns, K., Kalniņš, A.: Automatic construction of test sets: theoretical approach. In: Baltic Computer Science, pp. 286–359 (1991)
156. Schätz, B.: Verification of model transformations. In: Electronic Communications of the EASST, pp. 1–14 (2010)
157. France, R., Bruel, J., LarrondoPetrie, M.: An integrated object-oriented and formal modeling environment. Object-Oriented Program. **10**(7), 25 (1997)
158. Winkler, S., Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Softw. Syst. Model. (SoSyM) **9**(4), 529–565 (2010)
159. Bergmayr, A., Troya, J., Wimmer, M.: From out-place transformation evolution to in-place model patching. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, 647pp. –652 (2014)
160. Klatt, B.: Xpand: A closer look at the model2text transformation language. In: Language (2007)
161. Ráth, I., Bergmann, G., Ökrös, A., Varró, D.: Live model transformations driven by incremental pattern matching. In: Theory and Practice of Model Transformations, pp. 107–121 (2008)
162. Calisir, F., Calisir, F.: The relation of interface usability characteristics, perceived usefulness, and perceived ease of use to end-user satisfaction with enterprise resource planning (ERP) systems. Comput. Hum. Behav. **20**(4), 505–515 (2004)
163. Cho, V., Cheng, T.E., Lai, W.J.: The role of perceived user-interface design in continued usage intention of self-paced e-learning tools. Comput. Educ. **53**(2), 216–227 (2009)
164. Bastien, J.M.C., Scapin, D.L.: Evaluating a user interface with ergonomic criteria. Int. J. Hum. Comput. Interact. **7**(2), 105–121 (1995)
165. Kusel, A., Schönböck, J., Wimmer, M., Retschitzegger, W., Schwinger, W., Kappel, G.: Reality check for model transformation reuse: The ATL transformation zoo case study. In: AMT@MoDELS, pp. 1–11 (2013)
166. Louridas, P.: Version control software. In: IEEE Software, pp. 104–107 (2006)
167. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Softw. Syst. Model. **8**(1), 21–43 (2009)
168. Gardner, T., Griffin, C., Koehler, J., Hauser, R.: A review of OMG MOF 2.0 query/views/transformations submissions and recommendations towards the final standard. In: MetaModelling for MDA Workshop, vol. **13**, p. 41 (2003)
169. Abelein, U., Sharp, H., Paech, B.: Does involving users in software development really influence system success? In: IEEE Software, pp. 17–23 (2013)
170. Jackson, E. K., Schulte, W., Bjorner, N.: Detecting specification errors in declarative languages with constraints. In: International Conference on Model Driven Engineering Languages and Systems, pp. 399–414 (2012)
171. Kainz, G. G., Buckl, C., Knoll, A.: A generic approach simplifying model-to-model transformation chains. In: International Conference on Model Driven Engineering Languages and Systems, pp. 579–594 (2012)
172. Cuadrado, J. S., Guerra, E., de Lara, J.: Quick fixing ATL model transformations. In: International Conference on Model Driven Engineering Languages and Systems, pp .146–155 (2015)
173. Dubois, C., Famelis, M., Gogolla, M., Nobrega, L., Ober, I., Seidl, M., Völter, M.: Research questions for validation and verification in the context of model-based engineering. In: International Workshop on Model Driven Engineering, Verification and Validation (MoDeVVA), pp. 67–77 (2013)
174. Rivera, J. E., Guerra, E., de Lara, J., Vallecillo, A.: Analyzing rule-based behavioral semantics of visual modeling languages with Maude. In: Software Language Engineering, pp. 54–73 (2008)
175. Taentzer, G.: AGG: A graph transformation environment for modeling and validation of software. In: Lecture Notes in Computer Science, pp. 446–453 (2003)
176. EMFcompare https://www.eclipse.org/emf/compare/. Accessed 16 Feb 2018

177. Eclipse EMF query https://projects.eclipse.org/projects/modeling.emf.query. Accessed 16 Feb 2018
178. Diskin, Z., Gholizadeh, H., Wider, A., Czarnecki, K.: A three-dimensional taxonomy for bidirectional model synchronization. Syst. Softw. **111**, 298–322 (2016)
179. Varró, D., Asztalos, M., Bisztray, D., Boronat, A., Dang, D., Geiß, R., Greenyer, J., Gorp, P., Kniemeyer, O., Narayanan, A., Rencis, E., Weinell, E.: Transformation of UML models to CSP: A case study for graph transformation tools. In: Applications of Graph Transformations with Industrial Relevance, pp. 540–565 (2008)

Nafiseh Kahani is a Ph.D. Student in the School of Computing at Queen's University. She received her bachelor's degree in Software Engineering, and her master's degree in Information Technology. Her research areas of interest include model-driven engineering, software security, and real-time systems design.

Mojtaba Bagherzadeh is a Ph.D. candidate in the School of Computing at Queen's University in Canada. He obtained his Masters in E-commerce and Bachelors in Software Engineering. Mojtaba has several years of experience working as a software developer at IBM and leader at a startup company. His research areas of interest are real-time systems design, software engineering, and model-driven engineering.

James R. Cordy is Professor and past Director of the School of Computing at Queen's University in Kingston, Ontario, Canada, and Director of the NSERC CREATE Graduate Specialization in Ultra-Large Scale Software Systems. As leader of the TXL source transformation project with hundreds of academic and industrial users worldwide, he has been involved in computer software analysis and transformation systems for more than 30 years. Dr. Cordy is a registered professional engineer, a Senior Member of the IEEE, a Distinguished Scientist of the Association for Computing Machinery, and an IBM Visiting Scientist and Faculty Fellow. He has twice been recognized as IBM Centre for Advanced Studies Faculty Fellow of the Year, and most recently received the 2016 Queen's University Prize for Excellence in Research.

Juergen Dingel received an M.Sc. from Berlin University of Technology in Germany and a Ph.D. in Computer Science from Carnegie Mellon University (2000). He is Professor in the School of Computing at Queen's University where he leads the Modeling and Analysis in Software Engineering group. His research interests include model driven engineering, formal methods, and software engineering.

Daniel Varró is a full professor of software engineering and holds a research chair in Cyber-Physical Systems (CPS). His main research interest is model-driven software and systems engineering in the context of smart and safe CPS. He is a co-author of more than 100 scientific papers with five Distinguished Paper Awards, and two Most Influential Paper Award. He regularly serves on the program committee of various international conferences in the field, and he was a program committee co-chair of FASE 2013, ICMT 2014 and SLE 2016 conferences. He delivered a keynote talk at the IEEE CSMR 2012 and the SOFSEM 2016 conferences and at various international workshops and at the DSM-TP international summer school. He is a co-founder of the VIATRA model query and transformation framework and IncQuery Labs Ltd, an innovative Hungarian SME.