Comparison and Evaluation of Model Transformation Tools

Nafiseh Kahani and James R. Cordy

{Kahani, Cordy}@cs.queensu.ca

Technical Report 2015-627

School of Computing, Queen's University
Kingston, Ontario

December, 2015

Abstract

An impressive number of model transformation languages and associated tools have been developed over the last few years. These tools can be used to develop, transform, merge, exchange, compare and verify models or meta-models. In this paper, we compare and evaluate the current model transformation tools based on a qualitative framework. We begin with looking at the background areas of model transformation, and an overall taxonomy of current tools. We then classify, compare and evaluate the tools based on a number of facets, each one consisting of several attributes with the possibility of overlap.

Keywords: Model-driven engineering, model transformation tools, classification

1 Introduction

Model-driven engineering (MDE) is a rapidly expanding field that uses models as the fundamental elements in the entire process of software engineering. MDE techniques can simplify the design process, increase productivity and compatibility between systems, and boost the efficiency of the development process. Besides, MDE provides a comprehensive description of the system since various models can be used to describe different viewpoints. In this context, a system can be anything, such as a program or a computer system. A subset of MDE is model-driven development (MDD), a model-centric framework that uses models as the primary artifacts in the software development process. MDD extends the level of abstraction, while reducing the complexity of development. In MDD, abstract models are transformed into detailed models or code, so model transformations are essential. Model transformations can be compared to compilers in traditional programming languages.

A subset of MDD called Model-driven architecture (MDA) has been proposed by the Object Management Group (OMG) [78]. MDA uses OMG standards in the systems development process. The OMG have standardized the transformation definitions used in the MDA framework, by introducing the MOF model to text transformation language (MOFM2T) specification [79] for model-to-text (M2T), and the Query/Views/Transformation language (QVT) standard [80] for model-to-model (M2M) transformations. QVT consists of three languages namely, QVT Relational (QVTr), QVT Core (QVTc) and QVT Operational (QVTo) languages. There is also Architecture-driven modernization (ADM) that produces standards for model-based reverse engineering of legacy systems.

With MDD becoming more prevalent in software development, the number of model transformation techniques/tools has increased rapidly. There have been a number of publications [81, 82, 83, 84, 85, 86, 87, 88] classifying and comparing model transformation approaches and tools over different features. Some of these works have chosen a number of tools and compared them based on a limited range of attributes, but to our knowledge, there has not been a comprehensive comparison and evaluation of all of the current model transformation tools. In this survey, we catalogue all of the currently available model transformation tools, and compare them with respect to a range of attributes.

The remainder of this paper is organized as follows. Section 2 provides an overview of the research method we used in our study. Section 3 presents a basic introduction to model-driven background and terminology. Sections 4 and 5 classifies and compares tools based on a set of attributes. Section 6 evaluates tools according to a number of factors. Sections 7 and 8 discusses the results of previous sections. Section 9 examines the related work. In section 10, we conclude the paper.

2 Research Method

Our study followed the principles of a systematic literature and used three different sources to gather tools information. We began with the previously published MDD literature reviews, journal/workshop/conference papers in the field, specially articles related to the *Transformation Tool Contest (TTC)* to make a list of all of the existing model transformation tools. This step resulted in a list of 43 tools. Some of these have no website or download page (e.g., ArcStyler, Yet Another Transformation Language (YATL), Codagen Architect, OptimalJ, FUUT-je) or are still being developed, such as QVTd which is an implementation of

QVTc and QVTr [89], so we excluded them from the list.

To search further, we used web sites, such as *Google Scholar*, *Sourceforge*, and *Github* to find other tools not included in the existing survey papers. To do this, we searched tools based on some search keywords, such as "model transformation tools" and "model-to-model and model-to-text tools". We also attended the Models Conference (2015)[90] held in Ottawa to contact professors and students in the field to assure of the completeness of our work. In this paper, our focus is on the meta-model based modeling tools. However, during our systematic search we faced with tools, such as WebRatio which are not meta-model based, however fit very well in our taxonomy. Thus we consider them in our list as well. The final list includes 65 tools. We organize the current tools into a taxonomy based on transformation languages. Similar to previous work on clone detection tools [91] we perform a classification and overall comparison according to a number of facets, each of which has a set of attributes with the possibility of overlap.

3 Background

We begin with a basic introduction to model-driven terminology. Rothenberg et al. [92] 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, it is possible to perform different operations on them, such as merging to integrate models and produce a new model, or re-factoring to improve the internal structure of the model without changing its behavior or semantic.

Model transformation is a program used to transform a model from one representation to another. The input model of transformation is called source model which conforms to a source meta-model, and its output model so-called target model conforms to a target meta-model (in M2M transformations), or text (in M2T transformations). A model transformation description/specification written in a model transformation language, defines how one or more source model(s) are transformed to one or more target model(s). If the language of a transformation description is rule-based, the transformation description will consist of a set of transformation rules. The transformation engine/tool uses model transformation definition to produce the target model from the source model. Meanwhile, the models must be valid and expressed in a well-defined notation. Thus transformation specifications use the meta-model to define the appropriate and necessary structure, and properties to which a model must conform. There are models that define meta-models so-called meta-meta-models. This definition can continue recursively, so that meta-meta-models can be defined based on themselves. While in theory, there is any arbitrary number of meta-modeling levels, the OMG defined a four meta-modeling level architecture. In this standard, level $M0$ represents the real-world system, next level, $M1$ represents the modeling level of the system that is an instance of the next level. Level $M2$ is the meta-modeling level that describes the model in the level $M1$. The meta-meta-modeling in level $M3$ shows that meta-model conforms to itself. The relation between a model and its meta-model, and the meta-model with one of its models is shown with conformance and instantiation type respectively.

Modeling languages used to specify the models can be graphical, textual, or both. There are two classifications for modeling languages: domain-specific modeling languages (DSMLs) are dedicated to a particular domain or context for modeling purposes; and general-purpose modeling languages (GPMLs), such as the Unified Modeling Language (UML) [93], can be applied to any domain. DSMLs have relevant concepts to the domain, and thus support higher-level abstractions than GPMLs, which 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, properties and relations between the elements and validity constraints (i.e., the requirements for the model) of the model. Abstract syntax performs like a grammar for textual languages. In MDD, the abstract syntax is modeled by a meta-model. A concrete syntax can be textual to describe particular representations of models, graphical which uses graphical icons (e.g., tree-like) to show the elements of the model and relations among them, or both of them. Tools, such as Graphiti [94] and Xtext [95] can specify the graphical and textual concrete syntax respectively. 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 define the notation or the meaning of the relationship among language concepts. Thus, the semantics defined by denotational, operational, translational, and pragmatic approaches [96], is needed to describe the meaning of the different modeling elements and the different ways of combining them.

4 Classification of Tools

Based on the type of the target, model transformations tools can be classified into three main categories namely, model-to-model (M2M), model-to-text (M2T) and text-to-model (T2M) transformations. T2M transformation tools, e.g., MoDisco [97], accept text strings as input and models as output parameters of transformations. T2M tools are usually used for reverse engineering and need a parser generator, so we do not consider them in our work. Tables 1 and 2 provide a high-level overview of the tools based on a taxonomy of their transformation language. The third column in the table gives a sentence description of the tool, and the forth column shows the language in which it is implemented. *FR* and *LR* columns show the first and latest release of tools respectively. M2M and M2T transformation tools can be divided into the following subcategories.

4.1 Model-to-Model (M2M) Tools

M2M tools convert one or more source(s) into one or more target(s). Transformation languages provide a set of constructs or mechanisms to apply transformations. Relational, imperative, graph-based, others, and hybrid are different types of approaches to implement M2M transformation tools.

4.1 Relational/Declarative Approaches

These approaches focus on what should be transformed into what, without specifying an explicit sequence of execution order. Relational approaches are based on defining relationships between the elements in the source and target models. These relations are defined in mathematical relations, which can be specified by predicates and constraints. Relational approaches include functional programming, and logic programming. A logic language has many features, such as search mechanism, constraint propagation, and backtracking that make it appropriate to implement the relational approaches. In functional languages, a function similar to a transformation can transform the input(s) into the output(s). Object-oriented (OO) languages seem to be a straight approach for model transformations. However, functional language has the advantage that the developer does not need to deal with non-trivial task of writing code for model traversing. Tools such as UML-RSDS, Tefkat, JTL, PTL, ModTransf, PETE, and TXL are examples of relational approaches.

An example of the high-level relational model transformation approach is QVT Relation. In QVTr, e.g., Echo, MOMENT, QVTR-XSLT, ModelMorf, mediniQVT, a relation specifies by two or more domains with a pair of when and where clauses. Each domain represents a model in the transformation. The when clause determines the conditions under which the relationship needs to hold, and the where clause determines the condition that must be met by all model elements in the relation. Patterns are used to define the domains, which can be marked as check-only or enforced. In check-only mode, the consistency of the target model elements is checked, then based on the transformation the result can be true or false. In the case of false result, the rule is enforced by modifying the elements of target model, so that the target will be consistent. QVTr has both textual and graphical concrete syntax styles. Echo and mediniQVT tools are based on QVTr syntax, but using the semantics which is depart from the OMG standard.

QVT Core is a simple, low-level relational language based on pattern matching over a set of variables. The language evaluates conditions over those 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 QVc (act as the target of this mapping). QVTr is defined at a higher-level of abstraction, so it is more expressive and supports more complex pattern matching than QVTc.

4.1 Imperative/Operational/Constructive Approaches

Imperative languages focus on how and when the transformation should be executed, without taking into account the relations that must hold between source and target elements. The language specifies a transformation as a sequential actions/rules. Procedural languages such as MetaEdit+ are imperative and use procedures as abstraction mechanisms to wrap the sets of instructions. Imperative languages are similar to classic programming languages, so they are easy for developers to work with them.

QVT Operational is an example of imperative language, which is comparable to conventional procedural languages like C. In QVTo (e.g., QVTo-Eclipse, MagicDraw, OpenCanarias, SmartQVT, Together) transformations are defined using mappings. Each mapping can transform one or more element(s) of a source model to the corresponding target element(s). QVTo mappings, similar to relations in QVTr, may contain when and where clauses.

Table 1: Taxonomy of the M2M transformation tools

App.	Tool	Description	Lang.	FR	LR
Relational	UML-RSDS [1]	with verification support to construct software systems from UML spec.	Java	2005	2015
	Tefkat [2]	a rule-, pattern- and template-based engine implementation of Tefkat language	Java	2004	2008
	JTL[3]	specifically focuses on synchronization and change propagation models	ASP	2006	2015
	PTL [4]	ATL-style rules are combined with logic rules to define transformations	Java	2013	2013
	ModTransf [5]	a tool which its rules are expressed in XML	Java	2004	2005
	Echo [6]	used for model repair and transformations based on the model finder Alloy	Java	2013	2013
	MOMENT [7]	a partial support and editor for the QVTr	Java	2006	2007
	QVTR-XSLT [8]	based on the graphical notation of QVTr and XSLT	Java	2009	2009
	ModelMorf [9]	fully supports the QVTr language	Java	2006	2006
	mediniQVT [10]	uses QVTr language in the textual concrete syntax	Java	2007	2011
	PETE [12]	a Prolog rule-based tool that supports the transformation of Ecore models	Java	2009	2010
	TXL [65]	a grammar-based tool that can be used for model transformations	Turing+	1990	2015
Imperative	ModelAnt [13]	an extension of Apache ANT to support model transformations	Apache Ant	2004	2014
	Xtend[14]	the succr. of Xpand is a statically-typed high-level programming tool for JVM	Java	2013	2015
	MetaEdit+ [11]	a tool for domain-specific modeling and development	MERL	1993	2014
	QVTo-Eclipse[15]	an Eclipse implementation of Borland Together based on QVTo	Java	2008	2015
	Kermeta2[16]	based on a model-oriented language optimized for meta-models and DSLs	Java	2005	2012
	Modelio[17]	the succr. of Objecteering is based on UML and BPMN	Java	2009	2014
	Umple[18]	a programming language family to enable model-oriented programming	Java	2008	2015
	MDWorkbench[19]	Eclipse-based IDE for code generation and model transformations	Java	2005	2015
	Melange[20]	the succr. of Kermeta2 supports the semantics of the modeling languages	Java	2015	2015
	MagicDraw[21]	a visual UML, SysML, BPMN, and UPDM modeling tool	Java	1998	2015
	JAMDA[22]	creates Java code from a model of the business domain	Java	2002	2003
	Ente. Arch.[24]	a UML design and business analysis tool	C++	2000	2015
	OpenCanarias[26]	a virtual machine implementation of the QVTo mappings	Java	2008	2011
	SmartQVT[27]	a partial implementation of the QVTo language	Java	2006	2008
	SiTra[70]	a Java library for supporting a Java-based approach to implement M2M	Java	2006	2012
	WebRatio[71]	an application development platform for BPMN, WEBML, and IFML	Java	2001	2015
	Mitra2[73]	the succr. of Mitra optimized for semi-automated transformations	Java	2010	2012
	JQVT[74]	based on a compiled QVT engine for Java	Java	2012	2013
	Merlin[75]	based on EMF Java Emitter Templates (JET) templates and mapping model	Java	2004	2005
	Together[76]	a set of Eclipse plugins which partially implements the QVTo language	Java	2003	2015
	MOFScript[77]	the succr. of UMT which implements the OMG MOFM2T specification	Java	2006	2011
Graph-based	GROOVE[29]	a tool for model checking graph transformation systems	Java	2003	2014
	UMLX[30]	based on a concrete graphical syntax to complement the QVT language	Java	2005	2006
	AToM3[31]	a multi-paradigm modeling tool for visual languages	Python	2004	2008
	AToMPM [28]	the succr. of AToM3 generates domain-specific modeling web-based tools	Python	2012	2015
	AGG[33]	for the attributed graph grammar transf. which follows the algebraic approach	Java	1997	2015
	BOTL[34]	offers a protocol for the description of tool chains and model integration	Java	2003	2008
	GRoundTram[35]	a graph round-trip framework to develop bi-directional model transformations	OCaml	2009	2014
	eMoflon[36]	supports story-driven modeling and TGGs	Java	2006	2015
	Henshin[37]	the succr. of EMFTiger supports the visual modeling and transformations	Java	2011	2015
	MoTE[38]	provides bi-directionality,, model synchronization and model consistency	Java	2010	2015
	GReAT[40]	based on a pattern specification, a graph transf., and a control-flow languages	Visual C	2004	2014
	TGGInterpreter[41]	its rules are modeled as TGG-rule diagrams to provide transformations	Java	2006	2011
	MOMoT[69]	based on MDE and search-based optimization techniques	Java	2014	2015
	EMorF[42]	an incremental TGG tool which supports model synchronization	Java	2012	2012
	PROGRES[43]	an integrated environment and high-level language	C	1991	2004
	MoTMoT[51]	based on a UML implementation of story diagrams for graph rewriting	Java	2004	2006
Others	UMT[44]	based on UML/XMI which uses XSLT and Java for code generation	Java	2002	2005
Hybrid	VIATRA[45]	supports verif. to improve the quality of systems designed using the UML	Java	2000	2015
	Eclectic[46]	a tool based on family of model transformation languages	Java	2013	2013
	Epsilon[47]	a family of languages and tools with migration capability	Java	2006	2014
	AGE[48]	a tool based on the embedded DSLs, being Ruby the host language	Ruby	2006	2010
	VMTS[49]	a domain-specific meta-modeling and model processing framework	.Net C#	2003	2014
	ATL[50]	uses textual syntax and parts of the QVT specification to define transf. rules	Java	2005	2015
	Fujaba[63]	a story-driven modeling and graph transf. platform	Java	1994	2015
	GrGen.NET [67]	a programming productivity tool for graph transformations	Java	2003	2015
	Rational[23]	consists of a set of UML modeling tools for software design	-	1994	-
	Blu Age[25]	a set of plugins namely Reverse, Forward, Data and Data Base, and Analyzer	Java	2006	2015
	MOLA [39]	a graphical procedural transformation language	Java	2005	2014

There is another approach called direct manipulation approach (e.g., JAMDA, SiTra) which is similar to the imperative approach, but with lower-level constructs and language concepts to support model transformations. In this approach, general-purpose programming languages, such as Java, and VB along with the advanced capabilities, offered by Application Programming Interface (APIs) libraries, can be used to implement the model transformations. APIs enable users to create, manipulate or access the internal structure of models and meta-model instances using Java programming language. 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 have to manually implement many required features of model transformations, such as traceability. Furthermore, being dependent on particular APIs impose some restrictions on the type of transformations that the approach can support. Examples of imperative tools are ModelAnt, Xtend, Kermeta2, Modelio, Umple, MDWorkbench, Melange, Enterprise Architect (EA), WebRatio, Mitra2, JQVT, Merlin, and MOFScript.

4.1 Graph-based Approaches

Graph-based languages are based on algebraic graph grammars, and represent the source and target models using variations of typed, attributed, and labeled graphs. Graph transformation or graph rewriting consists of a set of rewriting rules (also called graph transformation rules or production rules), and a host graph to which the rules are applied to create a new graph. Each rule consists of left-hand side (LHS) graph, and right-hand side (RHS) graph. The execution of a graph rewriting rule on a host graph involves all elements that only are in the LHS are deleted, all elements that that are in RHS but not appearing in LHS are added, and all matched elements that exist in both sides remain unchanged. Optional negative application condition (NAC) is used to prevent the existence of certain patterns in the host graph, for instance the absence of particular vertices and edges. Most graph transformation approaches, such as GROOVE and AGG, allow specifying NAC for rules. Graph transformation rule containing a NAC is executed when a match for the LHS is found, and the NAC is not fulfilled. The act of trying to match a sub-graph is called pattern matching or evaluating a rule. The LHS and the NACs are the preconditions that must be satisfied prior to the rule execution, while the RHS is the post-condition of the graph rewriting rule. Graph transformations have solid theoretical foundation which allows to be used in formal verification of the transformations. The major drawback of graphical notation is the complexity and verbosity of representing the graph transformation rules. Examples of tools in this category are, AToMPM, GROOVE, UMLX, AToM3, AGG, BOTL, GRoundTram, GReAT, MOMoT, PROGRES, and MoTMoT.

Graph-based languages are suited to perform transformations between graph instances conforming to the same graph, therefore, they suffer from traceability difficulty between input and output graph instance elements. Triple Graph Grammars (TGG) [98] was proposed to overcome this disadvantage through using correspondence graphs or meta-models that maintain $N - to - N$ relation between source and target transformed 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. Taking modeling decisions by the developer and performing the manual changes on the models, make this approach almost difficult. TGGs are similar to QVTr but with strong theoretical basis. In QVTr the dependencies of transformation rules is explicitly formulated in the when and where clauses, while the order of TGGs rules implicitly specify based on the satisfaction of some preconditions. Examples of TGGs tools are eMoflon, Henshin, MoTE, TGG Interpreter, and EMorF.

4.1 Others Approaches

In this category, two approaches are mentioned by Czarnecki et al. [81], transformations implemented using xtensible Style-sheet Language Transformation (XSLT) and the application of meta-programming. Models can be represented and serialized as Extensible Markup Language (XML) using the XML Metadata Interchange (XMI) [99]. XSLT is a standard language for transforming XML that can be applied to implement model transformations. XSLT is a platform-independent, which uses the concept of patterns, and operates on textual representation of models. This approach traverses the XML tree structure to locate the nodes of the tree that match its pattern. In the case of pattern matching, XSLT applies a particular transformation rule. However, XSLT has scalability limitations, and its transformations are complex and verbose. Thus, maintenance of model transformations implemented in XSLT is almost difficult. In addition, XSLT can only support tree structures and not arbitrarily shaped graphs. Meta-programming approach proposed by Tratt [100] involves a domain-specific language (DSL) for model transformation embedded in a meta-programming language.

Table 2: Taxonomy of the M2T Tools

App.	Tool	Description	Lang.	FR	LR
Visitor	Kermeta2* [52]	based on a model-oriented language optimized for meta-models and DSLs	Java	2005	2012
	Melange*[53]	the succr. of Kermeta2 that supports the semantics of the modeling languages	Java	2015	2015
	JAMDA*[22]	creates Java code from a model of the business domain	Java	2002	2003
	ATOM3*[32]	a multi-paradigm modeling tool for visual languages	Python	2004	2008
	ATOMPM* [28]	the succr. of AToM3, generates domain-specific modeling web-based tools	Python	2012	2015
Template	ModelAnt* [13]	an extension of Apache ANT to support model transformations	Apache Ant	2004	2014
	ModTransf*[54]	a tool which its rules are expressed in XML	Java	2004	2005
	Umple*[18]	a programming language family to enable model-oriented programming	Java	2008	2015
	Acceleo[55]	a pragmatic implementation of the OMG MOFM2T standard	Java	2006	2012
	MagicDraw*[21]	a visual UML, SysML, BPMN, and UPDM modeling tool	Java	1998	2015
	AGE*[56]	based on the embedded DSLs, being Ruby the host language	Ruby	2006	2010
	eMoflon*[57]	supports story-driven modeling and TGGs	Java	2006	2015
	Henshin*[58]	the succr. of EMFTiger supports the visual modeling and transformations	Java	2011	2015
	MDWorkbench*[19]	a Eclipse-based IDE for code generation and model transformations	Java	2005	2015
	AndroMDA[59]	models created on UML tools can be transformed into deployable components	Java	2004	2014
	Fujaba*[64]	a story-driven modeling and graph transf. platform	Java	1994	2015
	TXL *[66]	a grammar-based tool that can be used for model transformations	Turing+	1990	2015
	WebRatio*[72]	an application development platform for BPMN, WEBML, and IFML	Java	2001	2015
	Ente. Arch.*[24]	a UML design and business analysis tool	C++	2000	2015
	UMT*[44]	a tool based on UML/XMI which uses XSLT and Java for generation	Java	2002	2005
	Merlin*[75]	based on EMF Java Emitter Templates (JET) templates and mapping model	Java	2004	2005
	MOFScriptt*[77]	the succr. of UMT implements the OMG MOFM2T specification	Java	2006	2011
	Rational*[23]	consists of a set of UML modeling tools for software design	-	1994	-
	Xpand[61]	is a the domain-specific M2T transformation framework for EMF models	Java	2004	2015
	Epsilon*[47]	a family of languages and tools with migration capability	Java	2006	2014
	VIATRA*[45]	supports verif. to improve the quality of systems designed using the UML	Java	2000	2015
Hybrid	Actifsource[60]	a domain-specific tool that generates running code from software specification	Java	2010	2015
	MetaEdit+*[11]	a tool for domain-specific modeling and development	MERL	1993	2014
	Blu Age*[25]	a set of plugins namely Reverse, Forward, Data and Data Base, and Analyzer	Java	2006	2015
	VMTS*[62]	a domain-specific meta-modeling and model processing framework	.Net C#	2003	2014
	Xtend*[14]	the succr. of Xpand is a statically-typed high-level programming tool for JVM	Java	2013	2015
	GrGen.NET* [68]	a programming productivity tool for graph transformation	Java	2003	2015
	Modelio*[17]	the succr. of Objecteering is a modeling tool based on UML and BPMN	Java	2009	2014

* (also M2M tool)

4.1 Hybrid Approaches

Each technique has its own strengths and weaknesses. In imperative approaches, a programmer has a high-level of control over the transformation execution which results in an efficient implementation of transformations, especially for complex ones. However, the explicit control can lead to writing more code that makes this approach harder to read and understand. As define transformations at a higher-level of abstraction, and hide the details related to the transformation process, relational languages make the task of model transformation development easier, concise and shorter. Being less expressive and providing less control for the developer, makes relational languages like XSLT approach not suitable for complex transformation tasks. Graph-based languages have also scalability problems to deal with large models. In this way, hybrid approaches, which combine the strengths of more than one model transformation approach, can be used to specify transformations. For instance, the developers can combine the graph-based approach with the imperative features (e.g., MOLA, VMTS, Fujaba, GrGen.NET) to design an intuitive and flexible transformation tool. VIATRA, Eclectic, Epsilon, AGE, ATL, Rational, and Blu Age are examples of hybrid transformation tools.

4.2 Model-to-Text (M2T) Tools

M2T transformation tools transform one or several model(s) into a stream of characters in terms of source code (e.g. C++, Java), or other textual forms, such as configuration files. Visitor-based, template-based, and hybrid are different types of transformation approaches to implement M2T transformation tools [81].

4.2 Visitor-based Approaches

Visitor-based approaches are similar to direct manipulation approaches, in a way that they also traverse a tree-based internal representation of a model to generate code for each model element. The generated code is written in a text stream. The order of models to be traversed, and what code to generate are defined by rules. However, the developer has to do some parts of the transformation, such as writing instructions in order to sending the text to the output. Examples of tools in this category are Kermeta2, Melange, JAMDA, ATOM3, and ATOMPM.

4.2 Template-based Approaches

Templates are the fundamental units of a template-based M2T language. A template defines the target text structures for the static part shared by all artifacts, and variables as matching model elements that can be replaced by values from the source model elements. There is a meta-program for the dynamic part, which provides accessibility to the stored information in the models. An example of this approach is MOFM2T that facilitates template compositions and module organization to handle complex M2T transformations.

As compare to visitor-based code generation, similarity of the structure of template to the generated code, makes this approach more accurate and easier to comprehend. In addition, the re-usability feature of templates makes the development process simpler. In programming language tools, such as GrGen.NET and TXL, the users can write their own visitors and templates. Examples of template-based tools are ModelAnt, ModTransf, Umple, Acceleo, MagicDraw, AGE, eMoflon, Henshin, MDWorkbench, AndroMDA, Fujaba, WebRatio, Enterprise Architect (EA), UMT, Merlin, MOFScriptt, Rational, Xpand, VIATRA, and Epsilon.

4.2 Hybrid Approaches

While visitor-based approach seems to be easier, it is not suitable when the most part of code generation consists of static text. Therefore, template-based languages can be combined with the visitor pattern to design and implement M2T tools. Actifsource, MetaEdit+, Blu Age, VMTS, Xtend, GrGen.NET, and Modelio are hybrid-based tools.

5 Comparison of Tools

In this section, we will describe the features of model transformation tools according to a systematic classification. At first we outline our classification scheme, and then classify and compare the tools using it. The properties are organized into facets, each of which may have different, but not necessarily separate attribute values. Related facets are grouped into categories. Tables 3 to 8 list the facets pertinent to each category. The second column in each table shows the full name of the facet, and the first column is the mnemonic abbreviation we use to refer to it. The third column is related to the unique identifiers of the facets attribute values. The last column provides short descriptions of the attribute values with the citations of some corresponding tools. Tables 9 and 10 assess the tools based on the mentioned facets.

5.1 General Category

The *General* category gathers facets relevant to the general usage of a tool (Table 3).

Update Time (UP): This facet shows the importance of accessing to the latest changes, and stable releases of the tool.

Operating Systems (OS) Platform: The facet describes the OS platform for which the tool is available.

Technological Platform (TP): A platform in general is a set of technologies or subsystems necessary to run the tool. The MDA guide [101] classifies platforms into generic platform types (batch, object), vendor specific platform types (Microsoft .NET, IBM WebSphere), and technology specific platform types (J2EE, CORBA). Examples of the used technological platforms in the tools are: J2EE in Blu Age, Together, WebRatio, and JaMDA; .Net in Blu Age and GrGen.NET; and J2SE in MagicDraw.

Availability (A): The *Availability* facet is concerned with the type of license under which the tool is accessible.

Available Resources (AR): The *Available Resources* facet is related to the up-to-date documentations (e.g., UML-RSDS, GrGen.NET, TXL, Actifsource), complete examples (e.g., ATL, MDWorkbench, GROOVE), forum (e.g., VIATRA, Umple, MetaEdit+), and so on that reflect the state of the tool. It is

Table 3: General facets

Abb.	Facet	Attr.	Description
UP	Update time	a	The tool is updated regularly; e.g., [1, 3, 65, 15, 17, 18, 21, 71, 76, 33, 36, 38, 60]
		b	The tool is updated sometimes; e.g., [11, 28, 29, 35, 41]
		c	The tool is updated never; e.g., [2, 5, 9, 12, 22, 74, 75, 34]
		d	Information not available; e.g., [26, 42]
OS	Operating System	a	The tool has been run on Windows; e.g., [39, 49]
		b	The tool has been run on Linux/Unix; e.g., [43]
		c	The tool has been run on Mac; e.g., [6, 12, 11, 17, 18, 28, 33, 34, 37, 19]
		d	The tool has been run on Windows, Linux/Unix and Mac; e.g., [20, 21, 71, 76, 38, 60]
		e	Information not available
TP	Technological Platforms	a	Vendor specific platform types; e.g., [49, 67]
		b	Technology specific platform types; e.g., [21, 22, 76]
		c	Both vendor and technology specific platform types; e.g., [25]
		d	No support
		e	Information not available; e.g., [26, 28, 33]
A	Availability	a	The tool is open source; e.g., [2, 6, 13, 15, 17, 18, 20, 73, 74, 75, 33, 34, 38, 46, 50, 67]
		b	The tool is freely available for research in binary form; e.g., [4, 65, 19, 39, 41, 60]
		c	The tool is commercially available; e.g., [11, 17, 19, 21, 24, 76, 60, 25]
		d	There is a free evaluation license; e.g., [11, 17, 19, 21, 76, 60]
		e	Information not available
AR	Available Resources	a	The tool provides documents (tutorial/user guide); e.g., [19, 22, 28, 33, 36, 37, 39]
		b	The tool provides examples; e.g., [15, 19, 28, 33, 36, 37, 38, 39]
		c	The tool has a wiki-page; e.g., [36, 37]
		d	The tool has a forum/community; e.g., [15]
		e	The tool has a website; e.g., [13, 19, 75, 28, 33, 36, 37, 38, 39]
		f	The tool has a download page; e.g., [13, 15, 19, 74, 75, 28, 33, 36, 37, 39]
		g	All of the above; e.g., [65, 14, 16, 17, 18, 21, 36, 37, 60, 50, 67]
EU	Ease of Use	a	The tool is similar to programming languages; e.g., [65, 14, 15, 17, 18, 19, 20, 35, 69]
		b	The tool is similar to script languages; e.g., [13, 17, 28, 33, 69]
		c	The tool is mathematical-/algebraic-based; e.g., [1, 34, 35, 36, 38, 39, 41]
		d	The tool is logic-based; e.g., [4, 12]
		e	All of the above
		f	Information not available
EM	Execution Environment	a	The tool is a plug-in for Eclipse; e.g., [2, 3, 11, 15, 16, 18, 19, 20, 21, 24, 73, 76, 37]
		b	The tool is integrated/dependent in other IDE; e.g., [11, 21, 24, 49]
		c	no IDE support; e.g., [9, 28, 33, 34, 35]
		d	The tool has a standalone APP; e.g., [2, 9, 13, 11, 16, 18, 21, 28, 29, 35, 38, 49, 63]
DA	Domain Application	a	The tool is a general tool; e.g., [65, 16, 19, 73, 36, 38, 39, 46, 48, 49]
		b	The tool can be used for web applications; e.g., [13, 28]
		c	The tool can be used for management information systems; e.g., [17]
		d	The tool can be used for real-time/embedded applications; e.g., [5, 17]
		e	Others/Information not available; e.g., [26, 74, 76, 34]
E	Extensibility	a	The tool supports extensibility; e.g., [3, 11, 15, 16, 17, 19, 28, 29, 36, 48, 49, 63, 67]
		b	no extensibility support; e.g., [1, 9, 65, 34, 38]
		c	Information not available; e.g., [20, 77, 35]
ED	External Dependencies	a	Possibly the tool has no external dependencies; e.g., [65, 11, 33, 35, 63]
		b	The tool seems to have external dependencies/to be a part of a larger tool set; e.g., [14, 15, 20, 74, 28, 36, 38, 39, 67, 25]
		c	Information not available
CS	Compatibility with Standards	a	The tool supports XMI standard; e.g., [65, 16, 19, 21, 76, 75, 29, 33, 37, 46, 48]
		b	The tool supports CWM standard; e.g., [2, 16]
		c	The tool is an implementation of QVTo; e.g., [15, 21, 26, 76, 49]
		d	The tool is an implementation of QVTr; e.g., [6, 9, 49]
		e	The tool is an implementation of QVTc; e.g., [76, 49]
		f	The tool is an implementation of QVT-Like; e.g., [2, 3, 74, 75, 34, 50, 25]
		g	The tool supports OCL expression; e.g., [3, 9, 16, 18, 19, 21, 76, 75, 37, 48, 50]
		h	The tool supports DD specification; e.g., [28, 33, 37, 25]
		i	The tool supports MOFM2T standard; e.g., [13, 77, 25]
		j	The tool supports HUTN standard; e.g., [28, 25]
		k	The tool supports JMI standard; e.g., [5, 13, 22]
		l	The tool supports CMI standard
		m	Information not available; e.g., [40]

important that the tool provides enough and complete resources so that users can easily understand how the tool works.

Ease of Use (EU): This facet specifies end user recognizability of the transformation syntax, and transformation definition. Some transformation languages are similar to programming (e.g., Mitra2, Kermeta2 is an OO language, Xtend is based on functional programming and OO language, QVTo-Eclipse is similar to procedural programming language), and scripting languages (e.g., Modelio, Tefkat has a syntax similar to Structured Query Language (SQL), ModelAnt uses scripting over an OO domain/wrapper of the model) which may have a smaller learning effort than graph-based (e.g., AToM3, graph query algebra UnCAL in GRoundTram) or logic-based (e.g., Prolog-based in PTL and PETE) tools. In these transformation tools, the beginners have to learn new material and development environments to work with the tool.

Execution Environment (EM): The *Execution Environment* facet captures whether the tool is part of an integrated development environment (IDE). Only a few tools support their own IDE, for instance, VMTS has its own IDE called VMTS Studio, and GReAT uses Generic Modeling Environment (GME) IDE. Advantages of the Eclipse platform, such as easily-extendable and being an unified platform makes it as a widely used underlying platform for model transformation tools, e.g., Xtend, eMoflon, and Henshin. Tools such as QVTR-XSLT that uses MagicDraw are dependent in other IDEs. Examples of tools that provide standalone APP are UML-RSDS, AToMPM, GRoundTram, AGE, Acceleo, Kermeta2 which provides a maven way to compile Kermeta programs to run as a plain Java APP, and GROOVE that runs standalone on the JVM.

Domain Application (DA): This facet shows the context to where the tool can be applied. MagicDraw can be used in systems engineering, business processes, enterprise architecture, and defense architecture areas. WebRatio is suited for mobile applications (class-platform); whereas Modelio can be used for enterprise architecture, software development, system architecture; and Actifsource is applicable for business-specific domains.

Extensibility (E): The *Extensibility* facet states whether it is possible to add new features and functionality to the tool, e.g., JTL, MDWorkbench, TGGInterpreter, AGE, Eclectic, Fujaba, GROOVE, ModTransf, Actifsource. MetaEdit+ has SOAP based API that allows the users to integrate other tools or develop their own functionalities. GrGen.NET is internally extended through programming new features by the users, and externally through using/calling libraries.

External Dependencies (ED): The facet indicates whether the tool requires additional other tools to work. Some tools are standalone, such as a transformation IDE, but depend on other tools. Some of the assessed tools have dependencies to other tools, such as Melange (used Xtext, and Kermeta 3 Action Language (K3AL)), Xtend (used Google Guava), ATOMPM (used Python-igraph and Chrome), Mitra2 (used Xtext), BOTL (used ArgoUML), eMoflon (used EA and ANTLR), MOTE (used MDELab Story Diagrams), MOMoT (used MOEA framework and Henshin), Blu Age (used MagicDraw), MOLA (used METAclipse tool), and VIATRA (used IncQuery and Xtend).

Compatibility with Standards (CS): In order to support the MDD approach in software development, we need standards to manage and transform models. OMG has developed a set of relevant standards that facilitate the use of MDA.

To improve interoperability between modeling tools, specific model interchange languages have been defined. XMI is a model interchange language for serializing and exchanging of models between modeling tools, and data repositories in a structured textual XML file. Different tool vendors adopt XMI with different versions. To overcome this problem, Canonical XMI [103] a constrained subset of XMI was introduced to minimize variability. Besides, eXtensible Graph Markup and Modeling Language (XGMML)[104] can be used to support graphical information in interchange process by XMI. The concrete syntax of XMI is excessively verbose and rather hardly readable by humans, thus OMG also issued Human usable textual notation (HUTN) [105] standard supported in AToMPM, Epsilon, and Blu Age.

The Common Warehouse Meta-model (CWM) [106] standard eases the interchange of warehouse and business intelligence meta-data between warehouse tools, warehouse platforms and warehouse meta-data repositories in distributed heterogeneous environments. CWM suffers from capability limitations, so it is not very common in model transformation tools. Moreover, the Object Constraint Language (OCL) [108] with a relational nature allows to write constraints on the meta-model, and validate queries on the model-level, e.g., used in PETE, TGGInterpreter, MoTE, ATL, and ModelMorf. ModelMorf uses OCL to specify templates, and when and where conditions in relations. Other languages with the capability of querying a model (e.g., Python in AToM3, Javascript in ATOMPM, Java in Fujaba and AGG, xBase in JQVT) can be used to define constraints. Constraints can define more precisely modeling languages, which leads to models with higher quality. To make understanding of OCL easier, Visual OCL (VOCL) [102] was introduced as graphical

visualization of OCL.

Tools such as AToM3, AToMPM, Blu Age, Henshin, and AGG supports Diagram Definition (DD) specification [107], which facilitates the definition of the mappings between the model elements, and their graphical notations. There is also MOFM2T standard (e.g., Acceleo, MOFScript) which considers language features, abstract and concrete syntax to specify M2T transformations. It is based on template approaches and OCL expressions.

Besides the OMG standards, the tools should be also able to provide support for legacy standards to facilitate the tasks, such as interoperability, and migration. ModTransf, ModelAnt, JAMDA, and Acceleo supports Java Meta-data Interface (JMI) [109] standard, which allows different UML tools to interact with each other through APIs.

There are other standards not mentioned in this category, such as Graph eXchange Language (GXL) [110] (e.g., used in GROOVE, AGG, GrGen.Net) which is a XML-based standard exchange format for graphs; Graph Transformation Exchange Language (GTXL) (e.g., used in AGG) that is an exchange format for graph transformations; Scalable Vector Graphics (SVG) (e.g., used in MetaEdit+); Interaction Flow Modeling Language (IFML) and Entity-Relationship model (ER) (e.g., used in WebRatio); DOT (e.g., used in GROOVE); Abstract Syntax Tree Meta-model (ASTM) and Knowledge Discovery Meta-model (KDM) (e.g., used in Blu Age); and UnQL (e.g., used in GroundTram).

5.2 Model-level Category

The *Model-level* category deals with modeling features of the tool. Table 4 summarizes these facets and their attribute values.

Modeling Languages (ML): Models need to be expressed in some modeling languages, which construct the source and target models of a transformation graphically or textually. UML is an OO modeling language, which consists of a set of different diagrams to visually describe the structure and/or behavior of a given system. UML provides a high-level of abstraction of defining models, however UML models do not have a formal semantics. Different tools can provide support for different UML diagrams, it depends on an application needs and practices to choose the UML elements and diagrams. For example; UML-RSDS supports UML2.x for class, use case, state machine, activity, sequence diagrams; Acceleo supports all UML2.x diagrams; MagicDraw provides full support for UML2.x and before UML2.x; PTL supports UML2.x for class diagrams; EA supports UML2.x for use case, activity, state, interaction overview, sequence, communication, package, class, object, composite, component and deployment diagrams; Blu Age supports UML2.x for class, activity, use case diagrams; ATOMPM, Umple, and Actifsource supports UML2.x for class and state diagrams; VMTS supports UML2.x for class, activity, use case, sequence, component, and deployment diagrams; ModelAnt supports before UML 2.x for class and state diagrams; Fujaba supports UML2.x for class, activity, and object diagrams; and Together supports UML2.x and before UML 2.x for class, activity, component, composite structure, deployment, state, use case, sequence, and communication diagrams. Moreover, some tools such as Kermeta2, Modelio, MagicDraw, EA, VIATRA, VMTS, and Acceleo use the Systems Modeling Language (SysML) [148] which is an example of UML profile. UML profile is an extension of the UML with additional semantic that allows to adapt the UML language for a particular purpose and area, through constraint and extension mechanisms. There is also the Executable UML (xUML) as a graphical specification language (e.g., VIATRA).

Petri net [117] processed with tools such as AToM3, AToMPM, VIATRA, and TGGInterpreter, is a graphical formal modeling language. It consists of places, transitions, and arcs, where the places are connected to transitions by input arcs and output arcs. In addition, programming languages such as OO programming can be used (e.g., in TXL, VIATRA, VMTS) to describe the models in a textual notation. Business Process Model and Notation (BPMN)[111] is also a graph-oriented standard for specifying business process modeling in a Business Process Diagram (BPD).

Some tools, such as MOTE and Tefkat, can process any model as long as it conforms to their metamodel. In addition, the assessed tools can process other modeling languages. For example, VMTS can process relational languages, and live script (imperative script language) which both are domain independent; GrGen.NET has its own model description language; and TXL supports Simulink. The supported modeling languages in Modelio are TOGAF, UPDM, SOAML, UTP, whereas EA supports BPEL, UPDM, TOGAF, SOAML, and SOMF.

Development Dimension (DD): This facet provides information about the abstract syntax, concrete syntax(es) and semantics of the tool. The majority of tools do not support the semantic aspect, whereas a

Table 4: Model-Level facets

Abb.	Facet	Attr.	Description
ML	Modeling Languages	a	UML 2.x; e.g., [4, 17, 18, 24, 71, 76, 28, 33, 37]
		b	Before UML 2.x; e.g., [5, 13, 22, 76, 33]
		c	xUML
		d	BPMN; e.g., [17, 24, 71, 76]
		e	Programming languages; e.g., [65, 76, 49, 25]
		f	SysML; e.g., [16, 17, 24, 49]
		g	Petri nets; e.g., [28]
		h	All of the above; e.g., [11, 38]
		i	Information not available; e.g., [15, 77, 40]
DD	Development Dimension	a	The tool has graphical concrete syntax; e.g., [21, 28, 29, 34, 38, 41]
		b	The tool has textual concrete syntax; e.g., [4, 6, 9, 65, 15, 16, 19, 20, 73, 50]
		c	The tool has both graphical and textual concrete syntax; e.g., [3, 5, 18, 35, 36, 49]
		d	Its abstract syntax/meta-modeling language is EMOF; e.g., [2, 5, 13, 34]
		e	Its abstract syntax/meta-modeling language is CMOF
		f	The tool supports both EMOF and CMOF meta-modeling languages
		g	Its abstract syntax/meta-modeling language is Ecore/EMF; e.g., [4, 6, 15, 19, 20, 73, 41]
		h	Its abstract syntax/meta-modeling language is KM3; e.g., [35]
		i	Other meta-modeling languages; e.g., [9, 65, 28, 39, 49]
		j	The tool support semantic of modeling language; e.g., [16, 20, 67]
		k	Information not available; e.g., [22]
LA	Level of Abstraction	a	The tool supports dynamic models; e.g., [1]
		b	The tool supports static models; e.g., [4, 9, 75]
		c	The tool supports both static and dynamic models; e.g., [17, 18, 21, 73, 33, 36]
		d	Information not available; e.g., [22, 77]
EXM	Execution Mode	a	The tool is interpreter-based; e.g., [3, 4, 6, 15, 29, 33, 41]
		b	The tool is compiler-based/code generator; e.g., [1, 13, 14, 17, 18, 19, 20, 21, 74, 76, 34, 39, 46, 63]
		c	The tool is both interpreter-based and compiler-based; e.g., [28, 36, 38, 69, 49, 67]
		d	Information not available
MH	Model Handlers	a	The tool supports EMF; e.g., [6, 12, 16, 19, 73, 76, 75, 37, 38, 39, 60, 46]
		b	The tool supports MDR; e.g., [5, 13, 20, 21]
		c	The tool supports both EMF and MDR; e.g., [55]
		d	The tool does not support model handlers; e.g., [74, 28, 33]
		e	Others/Information not available; e.g., [9, 11, 18, 39, 49]
MML	MDA Model-Levels	a	CIM; e.g., [11]
		b	PIM; e.g., [4, 5, 18, 73, 74, 75, 34, 36]
		c	PSM; e.g., [5, 18, 73, 75]
		d	All of the above; e.g., [15, 16, 19, 21, 24, 38, 41, 69, 63, 67, 25]
		e	Information not available; e.g., [77, 40]

large number of tools have concrete syntax as textual, graphical, or both textual and graphical, e.g., UML-RSDS, Umple, GRoundTram, eMoflon, and GROOVE. In GROOVE, parts of the production system must be edited graphically (rules and graphs), others textually (control, LTL/CTL properties and Prolog predicates). In Kermeta2, the semantic is defined by translational, and has an interpreted variant for validation purposes as operational semantic. GrGen.NET supports semantic through graph morphisms, category theory, denotational, operational, and pragmatic (a reference implementation) approaches.

There are several specific languages to define meta-models. For example, the Meta-Object Facility (MOF) [112] defined by the OMG is a standard to describe meta-models. MOF is divided into essential MOF (eMOF) and complete MOF (cMOF). eMOF is a simple framework based on a subset of UML class diagrams to define meta-models. cMOF provides more sophisticated features and graphical notation to specify complex modeling languages, such as UML. Ecore [113] proposed by Eclipse Modeling Framework (EMF) is another meta-model based on the eMOF specification. Kernel Meta-Meta-Model (KM3) [114] is a subset of Ecore to write meta-models in a textual representation. Using MOF or Ecore meta-modeling languages can increase the interoperability between MDD tools. Besides the mentioned meta-modeling languages, there are other meta-modeling languages supported by MDD tools. Some of these meta-modeling languages are GOPPRR (MetaEdit+), ArkM3 (ATOMPM), Genmodel (Henshin), MOLA MOF (MOLA), VPM (in previous version of VIATRA- before June 2015), VMTS Root (VMTS), and Umple (Umple).

Level of Abstraction (LA): Models can be used to describe the structure and behavior of the systems. Static models deal with the static structure of the objects in a system e.g., ER diagram, and UML class diagram. Dynamic models use the execution sequence of activities/tasks or information flows among different objects of the system to show the dynamic behavior of the system e.g., Petri net, UML activity or state diagrams.

Execution Mode (EXM): In order to generate a running system from the models, they need to be executable models. When the operational semantics of a model are completely defined, the model is complete enough to be executable, or the model is executable [118]. Strategies to implement execution tools, and thus make executable models execute are: code generation, like compiler, generates running code from a higher-level model to create a running application; model interpretation, like programming language, interpreters, parses and executes the model at run-time one statement at the time; and hybrid of both model interpretation and code generation (e.g., MOMoT, eMoflon, MoTE). In MoTE, TGGs are transformed into models of story diagrams, and then interpreted at run-time to perform the transformation. eMoflon is a hybrid approach in a way that uni-directional transformations and TGG rules are compiler-based, whereas TGG execution engine is interpreter-based. In general, GrGen.Net is compiler-oriented, but the rule application language is interpreted (unless used in the embedded rules). In JQVT, Java code is generated from a QVTr transformation. It does not need to re-interpret the transformation rule, thus JQVT can produce faster transformations which can be embedded into a Java application more easily than traditional QVT scripts could.

Model Handlers(MH): Models need to be stored and loaded to/from storages, such as files or repositories. Repositories used to store large models usually provide a particular API to access or manipulate the models. The EMF (e.g., used in JTL, PTL, PETE, MDWorkbench) is a Java modeling framework and code generation facility that based on a structured data model builds tools and other applications. Three fundamental pieces of EMF are, Core framework, EMF.Edit and EMF.Codegen. The Core includes the Ecore meta-model. EMF.Edit includes generic reusable libraries classes to build editors for EMF models. EMF.Codegen that provides code generation facility can generate everything needed to build a complete editor for an EMF model. In addition, some tools, e.g., Merlin, use an open source framework called Graphical Editing Framework (GEF) [115] along with EMF meta-models to create graphical editor for the existing application models on the Eclipse platform.

NetBeans Meta-data Repository (MDR) [116] as an example of JMI implementation is also a repository, which implements MOF. Meta-models and models can be imported into/exported from MDR using XML. The models are accessible through JMI API, or programmatically using the meta-model-specific. There are other model repositories, such as MetaEdit+ repository (MetaEdit+), JGraLab (MOLA), METADEPTH (Eclectic), VMTS repository (VMTS), GrGen.NET repository (GrGen.NET), MasterCraft (ModelMorf), and Umple repository (Umple). In some tools, e.g., AGE, Eclectic, VMTS, the model repositories can be extensible o other frameworks through an API.

MDA Model-Levels (MML): This facet notes three specific abstractions levels defined by the MDA namely, Computation-Independent Model (CIM), Platform-Independent Model (PIM) and Platform-Specific Model (PSM). CIM focuses on the domain, the specific requirements, and the purpose of the system, without

any binding to the details related to the targeted platform. PIM developed in accordance to the CIMs describes the system without the platform-specific details. PIM increases the level of abstraction via describing the behavior and structure of the system, independent from a particular platform. PSM contains all specifications and the required information of a particular type of platform (e.g., CORBA, .NET) integrated with the specifications in the PIM to determine how the system can use the platform.

5.3 Transformation Category

The *Transformation* category deals with the features of the transformation language the tool uses (Table 5).

Type (T): There is a distinction between endogenous and exogenous transformations based on the language in which the source and target models of a transformation are expressed [82]. Endogenous/homogeneous/ rephrasing transformations take place between models conforming to the same meta-model. Optimization and re-factoring of models are examples of endogenous transformations. Exogenous/heterogeneous/translation transformations take place between models expressed using different meta-models. Examples of exogenous transformations are refining models into more detailed models, such as code generation, reverse engineering, and migration of a model to a different platform. For instance, translation of UML model which is a platform-independent model into Java model as a platform-specific model.

Level (L): Vertical transformation changes the abstraction level of the model, an example is code generation [82]. Refinement transformation can increase the abstraction level, while an abstraction transformation reduces the amount of detail, so it decreases the abstraction level. Horizontal transformation just changes the representation of the model, whereas the source and target models remains at the same abstraction level. Re-factoring is as an example of horizontal transformation, so the abstraction level remains unchanged. Another example is language migration, where software model written in one programming language transform to another one, for instance, translating a UML class diagram to a ER diagram.

Direction (D): This facet shows that transformations can be uni-directional or bi/multi-directional. In uni-directional, transformations can only be executed from one particular source model to another particular target model. QVTo tools, e.g., QVTo-Eclipse, are an example that develop the model transformations in a way. In bi/multi-directional, usually relational tools, e.g., JTL, Echo, mediniQVT, ModelMorf, UML-RSDS, transformations can also be run in reverse or in multiple directions. Bi-directional transformations can be determined by dividing transformation mode into forward (source-to-target) and backward (target-to-source) transformation specified by two uni-directional transformations. It is also possible to use a transformation language that supports definitions of bi-directional transformations where every program describes both a forward and a backward transformation simultaneously. The advantage of using the bi-directional transformation is to check the consistency between models when changes on the target model are allowed. Bi/multi-directional transformations can also be used for reverse and round-trip engineering, software evolution, and synchronization to keep two or more models consistent, through reflecting the changes from one model to the other one(s). TGGs tools, e.g., eMoflon, TGGInterpreter, MoTE, EMorF, are also well-known examples of bi-directional model transformations. Other tools such as BOTL, and GRoundTram support bi-directional transformations, where based on the rules, it can be ensured for valid source models valid target models are generated.

Scope (S): This facet indicates that a set of mappings/transformations between different MDA model types (PIM, PSM, and CIM) should be supported by the tools. PIM-to-PSM is an example of a vertical transformation which allows to drive many PSMs from a single PIM. Some of the assessed tools, e.g., MOLA, TXL, Kermeta2, Modelio, Melange, AGG, ATL, Fujaba, can be used for any transformations between such models. In fact, the MDA classification is ignored as long as all relevant models are correctly specified by their meta-models. It is also possible to generate CIM-to-Code (e.g., Acceleo, MetaEdit+), PIM-to-Code (e.g., ModelAnt, MetaEdit+, Acceleo), or Code-to-PIM (e.g., MetaEdit+, Acceleo). These transformations do not explicitly model any platform-specific features, conventions, or requirements. In ModelAnt, the models can be annotated with tagged values to drive the platform-specific code generation. Most domain-specific language generators are best characterized as CIM-to-Code or PIM-to-Code.

Cardinality (C): The *Cardinality* facet classifies model transformations based on the number of input and output models. In ModelMorf, relations among N models can be specified, but at a time the user can only transform one model to enforce the specified relations. MOLA can process only one input model in the chosen repository to create one target model (i.e., 1-to-1). However, the physical model can contain many logical models. The target model similar to meta-model can be split into logical fragments using packages.

Table 5: Transformation facets

Abb.	Facet	Attr.	Description
T	Type	a	Exogenous transformations; e.g., [6, 74, 75, 60, 55]
		b	Endogenous transformations; e.g., [29, 33, 34]
		c	Both exogenous and endogenous transformations; e.g., [1, 15, 24, 73, 48, 49, 50, 67]
		d	Information not available
L	Level	a	Vertical transformations; e.g., [11, 60, 55]
		b	Horizontal transformations; e.g., [74, 29, 34, 38, 39]
		c	Both vertical and horizontal transformations; e.g., [5, 14, 11, 16, 18, 19, 21, 48, 49]
		d	Information not available; e.g., [40]
D	Direction	a	Multi-directional transformations
		b	Bi-directional transformation; e.g., [1, 3, 6, 35, 36, 38]
		c	Uni-directional transformation; e.g., [2, 6, 65, 19, 20, 21, 24, 26, 33, 35, 36, 38, 50]
		d	All of the above; e.g., [9, 41]
		e	Information not available
S	Scope	a	CIM-CIM (computational independent models can be refine); e.g., [15]
		b	CIM-PIM (computational independent models are transformed into platform-independent models); e.g., [15]
		c	PIM-PIM (abstract or refine models without binding to any platform-specific information); e.g., [4, 9, 15, 18, 73, 74, 29, 35, 36]
		d	PIM-PSM (a platform-independent model, with enough transformation definitions can be transformed to a platform-specific model); e.g., [1, 4, 9, 15, 18, 73, 35, 36]
		e	PSM-PIM (use for reverse engineering and are rather difficult to drive); e.g., [3, 18, 36]
		f	PSM-PSM (abstract or refine platform-specific models during the component realization and deployment); e.g., [9, 15, 18, 73, 74, 75, 35, 36]
		g	PSM-Code (PSMs are translated into software artifacts); e.g., [18, 75, 36, 55]
		h	Code-PSM (used for reverse engineering); e.g., [18, 36, 55]
		i	All of the above; e.g., [11, 16, 19, 33, 38, 49, 63]
		j	Information not available
C	Cardinality	a	1-to-1 (there is one source model and one target model); e.g., [6, 28, 29, 34, 35, 38]
		b	1-to-N (can produce several target models e.g., model merging); e.g., [18, 74, 55, 49]
		c	N-to-1 (several source models are combined into a single model)
		d	N-to-N (one or more input model(s) is transformed into one or more target model(s)); e.g., [5, 18]
		e	All of the above; e.g., [1, 2, 3, 14, 11, 15, 16, 17, 19, 76, 33, 36, 41, 46, 50]
		f	Information not available
RS	Rule Scheduling	a	Its form is sequential/explicitly; e.g., [15, 19, 20, 73, 74, 75, 28, 35, 36, 39, 49]
		b	Its form is not-sequential/implicitly; e.g., [2, 71, 33, 34, 38, 48]
		c	The tool supports both implicit and explicit forms; e.g., [65, 46]
		d	Its rule selection is explicit condition; e.g., [4, 15, 19, 20, 71, 73, 74, 75, 39]
		e	Its rule selection is non-determinism; e.g., [28, 33, 34, 38, 41]
		f	Its rule selection is conflict resolution; e.g., [18, 71, 63]
		g	Its rule selection is Interactive; e.g., [73, 28, 33]
		h	All the above rule selection mechanisms; e.g., [36]
		i	Its rule iteration is recursion-oriented; e.g., [2, 4, 9, 16, 20, 74, 75]
		j	Its rule iteration is looping-oriented; e.g., [14, 16, 19, 20, 28, 36, 39, 41]
		k	Its rule iteration is fix-point-oriented; e.g., [9, 33, 34, 36, 38]
		l	All the above rule iteration; e.g., [49]
		m	The tool supports phasing; e.g., [15]
		n	All of the above; e.g., [29, 25]
		o	No support; e.g., [17]
		p	Information not available; e.g., [76]
RO	Rule Organization	a	The tool supports modularity; e.g., [1, 3, 16, 20, 71, 73, 36, 39, 69]
		b	Its reuse technique is inheritance; e.g., [1, 9, 16, 18, 19, 74, 33, 36]
		c	Its reuse technique is logical composition; e.g., [2, 9, 65, 33, 35, 38, 69]
		d	All of the above; e.g., [14, 15, 46, 63, 25]
		e	No support; e.g., [17, 28]
		f	Information not available; e.g., [76]
RAC	Rule Application Control	a	Its rule application control is deterministic; e.g., [1, 2, 4, 16, 18, 19, 20, 71, 73, 74]
		b	Its rule application control is non-deterministic/concurrent; e.g., [28, 34, 39, 63]
		c	Its rule application control is non-deterministic/one-point; e.g., [33, 41, 69, 63]
		d	Its rule application control is interactive; e.g., [71, 73, 28, 33, 49]
		e	All of the above; e.g., [29, 36, 67, 25]
		f	No support; e.g., [17]
		g	Information not available; e.g., [76]

Rule Scheduling (RS): A large number of currently model transformation tools are based on transformation rules. Each rule describes how the source model to target model transformation is specified, in particular with respect to implementation of transformation rules. Common examples of transformation rules are rewrite rules with a LHS and a RHS. Transformation rules can be implemented by general programming languages, such as Java, or graph transformation languages, such as TGGs approaches.

The form of scheduling mechanisms determine the way in which individual rules are applied and can vary in four main areas namely, form, rule selection, rule iteration, and phasing [81]. The form can be expressed implicitly or explicitly. In implicit, the transformation engine selects the execution order based on implicit relations among the rules. This mechanism is common in relational model transformation tools, e.g.,Tefkat, JTL, PTL, PETE, other examples are binding-based in ATL, and pre/post-conditions in QVTr. The execution orders of rules can be changed by the patterns and connections between the rules designed by the developer. In explicit style, the user has direct control over selecting the execution of rules using some dedicated features of the transformation language, such as loops and conditionals. This style is usually found in imperative tools, e.g., Xtend, QVTo-Eclipse, Umple, MDWorkbench, Melange, JAMDA, WebRatio, JQVT, Merlin, ModelAnt, Kermeta2, MagicDraw, and hybrid model transformation tools, e.g., Fujaba, VMTS, which control flow is typically user-defined. There are different ways to control a structure, for example using story diagrams based on UML activity diagrams in VMTS, abstract state machines (ASMs) language in VIATRA, state-charts and activity diagrams in Fujaba, rule priorities in ATOM3, and transformation units in Henshin. Although this form provides full control over the transformation execution, in complex scenarios, the developer has to do the most of the work. On the other hand, in a relational transformation, the developer does not have control on the most tasks of a model transformation that makes these approaches obscure and hard to comprehend. Therefore, the transformation language with both implicit and explicit rule scheduling are more comprehensible and flexible as in ATL, Mitra2, GROOVE, Epsilon, Eclectic, UML-RSDS, TXL, Xpand.

Rule selection can be deterministic, non-deterministic, conflict resolution, or interactive [81]. In deterministic, an algorithm control the order of application of rules. It is possible that exists more than one applicable rule to the same part of the source model. In this case, the order of application of rules may be non-deterministic, for different executions of the same transformation on the same source model. In addition, different transformation executions could lead to different results, so there is a probability of conflicts and infinite recursion, In the case of *rule conflict*, conflict resolution strategies like explicit priorities or layers can be used. Rules or sets of rules organized into layers are executed in a certain order, so that rules in a layer do not cause conflicts with rules in the other layers. In explicit priorities, the rule with the highest/lowest priority is evaluated first. Only when rules with a higher priority fail to match, rules with a lower priority can be evaluated. In interactive, the user can be involved in deciding how different transformation rules can be scheduled. Interactive selection are supported in AToMPM, Xtend, AToM3, Mitra2, and GrGen.NET.

Rule iteration mechanisms include recursion, looping, fix-point iteration (i.e., repeated application until no changes detected), and a combination of them [81]. In MOMoT, rule scheduling is derived based on quality of models, similar to fix-point based on the quality of output models. Phasing, e.g., found in QVTo-Eclipse, AGE, PETE, ModelMorf, Tefkat, also means that the process of transformation can be organized into a sequence of phases, where each phase has a specific purpose and each phase has its own set of specific rules.

Rule Organization (RO): This facet determines the organization of transformation rules [81]. Modularity mechanisms group rules into modules (e.g., JTL, Acceleo, VIATRA), while reuse mechanisms allow to define a rule based on one or more existing rule(s), such as using inheritance between rules, or composition of transformation rules to avoid code duplication and consequently maintenance problems. Tools such as AGG, Blu Age, Fujaba, Eclectic, Xtend, QVTo-Eclipse, and ModelMorf that provide language composition and inheritance features can ease the process of model transformation. QVTo-Eclipse supports logical composition in terms of disjucting and merging mappings. In ModelMorf a rule can be composed from other rules, through invoking them in its where clause.

Rule Application Contro (RAC): This facet is related to the mechanisms that determine the rule application location(s) within a given source scope. The strategy can be deterministic, non-deterministic or interactive [81]. The non-deterministic strategies are divided in concurrent, and one-point. If a rule is possible to be applied at several matched locations concurrently, non-deterministic strategy with concurrent application can be used. If the transformation is non-deterministic, it may result in more than one way to keep models consistent. In one-point application a rule is applied to one non-deterministically selected location. In interactive strategy, the user decides the location to where a rule to apply.

Table 6: Capability facets

Abb.	Facet	Attr.	Description
V	Verification	a	Correctness for syntactic and semantic (the correct models of source language result in the correct models of target language); e.g., [1, 3, 29, 33, 34, 36, 38, 41, 49]
		b	Termination (a transformation always stops executing after a finite number of steps and leads to a result); e.g., [1, 29, 33, 34, 38, 49]
		c	Consistency (models are consistent with each other); e.g., [3, 6, 9, 18, 33, 34, 36, 38, 41]
		d	Completeness (A forward transformation is called complete if each element of the source model can be transformed to an element of the target model, and vice versa); e.g., [1, 34, 38]
		e	Determinism/Uniqueness/Confluence (different executions of the transformation always produce the same result); e.g., [1, 29, 33, 34, 38]
		f	Comprehensibility (the developed model is comprehensible by the user(s)); e.g., [67]
		g	Robustness (the ability to manage invalid models); e.g., [6, 18, 71, 29, 33]
		h	Definedness (the transformation can be applied to every model of the source language); e.g., [1, 6, 38]
		i	All of the above
		j	No support; e.g., [65, 15, 24, 74, 75, 28]
		k	Information not available
VA	Validation	a	The tool provides a testing environment; e.g., [16, 17, 19, 60, 25]
		b	The tool provides a simulation environment; e.g., [29, 33, 34, 49]
		c	Both simulation and testing environments; e.g., [14, 18, 24, 67]
		d	No support; e.g., [12, 65, 13, 15, 75, 55]
		e	Information not available
I	Input	a	User-defined models (created manually by the user(s)); e.g., [18, 19, 73, 75, 28, 38]
		b	Derived models (created automatically by the program(s))
		c	Both user-defined and derived models; e.g., [25]
		d	Information not available
O	Output	a	In-place; e.g., [1, 15, 16, 17, 18, 19, 20, 24, 73, 75, 28, 29, 33, 35, 36, 37, 39]
		b	Out-place; e.g., [1, 6, 11, 15, 16, 17, 18, 19, 20, 24, 73, 74, 75, 33, 35, 36, 37, 39, 46]
		c	Textual artifacts; e.g., [11, 16, 17, 18, 19, 20, 36, 69, 60, 55]
		d	Source code; e.g., [13, 11, 17, 18, 24, 75, 37, 60, 55]
		e	Database artifacts; e.g., [17, 19]
		f	Query; e.g., [13, 28, 35, 55, 50]
		g	All of the above; e.g., [14, 25]
		h	Information not available; e.g., [27, 40]
ET	Editing Tasks	a	Access transformations; e.g., [74, 48]
		b	Add transformations; e.g., [74, 48]
		c	Update transformations
		d	Delete transformations; e.g., [74, 48]
		e	All of the above; e.g., [6, 65, 16, 19, 20, 24, 71, 73, 76, 29, 36, 37, 69, 60, 50]
		f	Information not available; e.g., [42]
MP	Meta-Programming	a	A meta-programming tool; e.g., [11, 16, 20, 28, 31, 49]
		b	No support; e.g., [12, 15, 18, 75, 35, 36, 38, 55]
		c	Information not available
RE	Reverse Engineering	a	A reverse engineering tool; e.g., [13, 11, 17, 18, 19, 24, 71, 76, 55, 25]
		b	No support; e.g., [12, 14, 15, 20, 75, 28, 35, 36, 38, 39, 67]
		c	Information not available
RT	Round-trip Engineering	a	A round-trip engineering tool; e.g., [17, 24, 76, 35, 38, 25]
		b	No support; e.g., [12, 15, 16, 18, 20, 71, 28, 33, 36, 39, 41, 48, 67]
		c	Information not available

5.4 Capability Category

The *Capability* category groups facets that characterize the kinds of engineering features the tool is able to provide (Table 6).

Verification (V): Verification and validation are the process of determining whether a system satisfies desirable properties, such as termination or determinism. Formal methods and testing approaches are two options to validate the behavior of the model transformations. Formal verification can be applied to validate the transformation based on a complete input space or just a subset of input models. The second approach may not be complete and precise but it is more efficient for large transformations.

MDD has a teamwork nature so models may be developed by independent teams of designers. In this way, models can be inconsistent with the meta-models or other coexisting models. Thus, consistency between multiple related models are a crucial issue. Consistency can be check by bi-directional or incremental model transformations, model synchronization, or the constraint languages such as OCL. Tools, such as Echo can be able to find and repair inconsistencies. In ModelMorf when a transformation is executed in check-only mode, the models specified by the relations are checked for the mutual consistency. Another important property is completeness which is usually related to both individual rules and the entire transformation [119]. Notice here that some tools just focus on the well-formedenees property. However, well-formedenees is usually just considered for a single model, thus does not guarantee the correctness of a model or consistency between several complementary models.

Termination guarantees the existence of target model(s) through avoiding constructs that can be applied indefinitely to the target models. Model transformation approaches are usually Turing-complete while termination refers to Turings halting problem which is known to be un-decidable. There are approaches to define sufficient termination conditions for model transformation systems [120, 121, 122].

In the case of small models, simplicity and readability of models can help the user(s) to verify the models themselves. Mohagheghi et al. [123] show that understandability by intended users can be improved by well-organized models. Furthermore, it is so important that the transformation can continue its execution in the case of occurring errors during the execution, or work with every model of the source language. The majority of MDD tools do not provide built-in verification and validation functions. In tools, such as Kermeta2 and MetaEdit+, users can manipulate invalid models and make them valid, or build a checker that makes sure only valid models will be handled.

Validation (VA): Testing executes a model transformation on input test models and checks and validates its behavior. Passing a test alone cannot fully verify that the system's implementation behaves as expected under all conditions. Nevertheless, a test case with the adequate test inputs can be an easy and feasible strategy for validation of large and complex model transformations. There are three main approaches in testing namely, *white-box*, *black-box* and *grey-box*. In white-box test, a test suite is designed based on the full implementation of the transformation. In black-box, the implementation of the transformation is not available, while in grey-box the implementation of the transformation is partially available. The main drawback of testing is that the exhaustive testing is time-consuming and not always possible. Tools such as Kermeta2, Modelio, Actifsource, VIATRA, PTL, MDWorkbench, and Fujaba provide a testing environment. For example, Fujaba uses graph-transformation-based JUnit tests. Simulation of the system also allows designers to model and simulate behavior of the model. The formal verification and validation of model transformations has increasingly been a topic of research in recent years [126, 127, 128, 129, 130, 131, 143, 149].

M2M transformations are executed between models, therefore any reference to the updated model elements or elements that do not exist anymore makes a problem. Furthermore, M2M transformations defined at the meta-model level are dependent to changes or evolution of meta-models. In this case, model migration is needed to keep a model conform to its corresponding meta-model evolution. Tools, such as Edapt [124] and Epsilon Flock [125] can support model migration. Technology is constantly changing, so platform evolution also impacts M2T transformations. Therefore, any change must be reflected in the model transformation.

Input (I): The *Input* facet captures the type of input as user-defined or derived models supported by a particular tool.

Output (O): The *Output* facet indicates the kind of output supported by a particular tool. In-place/destructive transformations take only one model, where the source and target model are the same. In this case, the target model is created by directly recreating, deleting, and updating elements of an existing source model. Model refinements, model re-factoring, optimizations are examples of in-place transformations, where the elements of the model that remain unchanged by the transformation are not necessary to be copied in the target. Meanwhile, It is possible to emulate in-place transformations by copying the source model to the target model and modifying it. In-place transformations are suitable for endogenous transformations,

and normally implemented using graph-based approaches as in GROOVE, AToMPM, GRoundTram, and MOMoT. To ensure termination property in approaches with the non-deterministic selection and fix-point iteration scheduling, in-place update may be limited.

There is also out-place/conservative transformations which generate a new target model from the scratch. Out-place transformations are suitable for exogenous transformations. It is also possible to emulate the out-place transformations, e.g., in AGG, because the user can model two different languages in one integrated model.

Query as another type of output, is performed on the source models to request specific contents or a selection of the model elements. A relational query describes the relations between variables/entities, independent of the details implementation. While an imperative query binds with the manipulation. If a query only return elements from the queried model, it is called selective, otherwise constructive.

Database and textual artifacts are other types of transformation output. TXL has been used to transform models in XMI form to Prolog facts and SQL tables. MagicDraw can produce textual artifacts as template files, such as plain text, RTF, HTML, XML template, and database structure without data. Umple can produce documentations in HTML and JavaDoc formats, and analysis reports. Source code is also another type of output, for example, MetaEdit+, Xtend, ModelAnt, VMTS and Actifsource can generate any source code. Acceleo generates JavaEE, C#, Python, Zope, PHP; Modelio generates Java, C++, C#, SQL; EA generates C++, C#, Java, Delphi, VB, SQL; VIATRA generates Java, C++; AGE, Henshin, Fujaba, Merlin, ModTransf, and WebRatio generate Java; MagicDraw generates Java, C#, C++ ; Blu Age generates JEE, .Net, JavaScript; Umple generates Java, C++, Ruby, PHP; and TXL generates C programming language.

Editing Tasks (ET): This facet is related to creating new transformations or modifying the existing ones by applying editing operations, such as accessing models elements/properties, adding/removing model elements, or updating elements values. Some tools are not able to update the values of model elements from an existing model. For example, AGE is not able to modify model elements in in-place transformations. The user can create a copy of the model and then update/remove its elements while copying.

Meta-Programming (MP): This facet indicates whether the tool is intended for meta-modeling, such as Melange, MetaEdit+, Kermeta2, AToM3, AToMPM, VIATRA, and VMTS.

Reverse Engineering (RE): Some tools, e.g., MDWorkbench, ModelAnt, MetaEdit+, Modelio, Umple, EA,Blu Age, Fujaba, WebRatio, can extract higher-level specifications from lower-level ones, in terms of generating models from the code. In bi-directional transformations, unlike reverse engineering, the same model transformation language applies for both forward and backward transformations.

Round-trip Engineering (RT): This facet indicates that the tool can create models from source code and vice versa. Tools should support both creating source code from platform-specific models, and platform-specific models from source code.

5.5 Implementation Category

The *Implementation* category relates to the usability of the tool (Table 7). The richer this features, the further processing the tool can provide.

Editor (EDI): This facet describes whether the tool is visual or command-line/textual. Some tools, e.g., Umple, ATOMPM, provide the access to the tool through a web APP.

Workspace and Project Management (WPM): This facet indicates, whether the tool provides workspace to make the easy management of the resources, such as projects, or individual files.

Teamwork Support (TS): This facet indicates, whether the tool supports team and collaboration activities. In modeling, team members need to discuss on models or collaborate on the shared models simultaneously. Nowadays, there is a trend of using cloud computing to handle models distributed among different machines [145].

Syntax Editor (SE): This facet supports the facilities such as syntax highlighting, auto formatting, code completion, code navigation, and code folding that allow users to work easily with the tool. Some tools use editing features proposed by other tools, for example Echo inherited these features from QVTs Eclipse editor, and editing features in Mitra2 supported by Xtext.

Semantic Editor (SYE): This facet provides appropriate support for the user to determine and fix the origin of failure(s). Features related to this facet are, re-factoring, error and warning detection, quick fixes, debugger, reference resolution, build systems, and profiler.

Re-usability Technique (RUT): Reusing tested sequences/rules/functions/procedures/patterns/ transformations can boost the scalability, efficiency, and quality of the transformation tools. Composition, orchestration, decomposition, generic, and higher order transformations (HOTs) are examples of re-usability

Table 7: Implementation facets

Abb.	Facet	Attr.	Description
EDI	Editor	a	Graphical; e.g., [4, 13, 15, 16, 17, 73, 74, 76, 75, 39]
		b	Command-line; e.g., [65, 46]
		c	Both graphical and command-line; e.g., [6, 11, 18, 29, 33, 35, 36]
		d	Information not available; e.g., [42]
WPM	Workspace and Proj. Mngmt	a	The tool has workspace and project management; e.g., [2, 16, 17, 18, 24, 75, 36, 39, 50]
		b	No support; e.g., [69]
		c	Information not available
TS	Teamwork Support	a	The tools is multi-users; e.g., [36]
		b	The tool is multi-projects; e.g., [15, 49]
		c	The tool is both multi-users and multi-projects; e.g., [14, 17, 18, 19, 28, 60, 46, 48, 25]
		d	No explicit teamwork support; e.g., [5, 74, 76, 75, 29, 38, 39, 41]
		e	Information not available; e.g., [20]
SYE	Syntax Editor	a	The tool has syntax highlighting; e.g., [18, 74, 36]
		b	The tool has auto formatting
		c	The tool has code completion; e.g., [74, 36]
		d	The tool has code navigation; e.g., [2, 15]
		e	The tool has code folding; e.g., [15, 18]
		f	All of the above; e.g., [14, 15, 17, 19, 20, 76, 69, 60, 55, 25]
		g	No support; e.g., [4, 13, 28, 38, 41]
		h	Information not available; e.g., [42]
SE	Semantic Editor	a	The tool has re-factoring; e.g., [16, 39]
		b	The tool has error and warning detection; e.g., [15, 16, 18, 19, 74, 28, 29, 35, 38, 48]
		c	The tool has quick fixes; e.g., [74, 55]
		d	The tool has debugger; e.g., [3, 4, 15, 16, 18, 19, 28, 29, 35]
		e	The tool has reference resolution; e.g., [15, 46]
		f	The tool has build systems; e.g., [2, 16, 46, 48]
		g	The tool has profiler; e.g., [15, 55]
		h	All of the above; e.g., [14, 17, 60, 25]
		i	No support
		j	Information not available; e.g., [27]
RUT	Re-usability Technique	a	Composition (compose the existing parts/transformations to construct a new or more complex transformation(s)); e.g., [1, 9, 12, 65, 18, 20, 73, 28, 39, 69, 46]
		b	Orchestration (use the whole transformation at once); e.g., [14, 28, 69, 46]
		c	Decomposition (split a complex transformation into appropriate and separate small components); e.g., [12, 65, 20, 46]
		d	Generic; e.g., [14, 20]
		e	HOT (takes as input a transformation model and generates a transformation model as output); e.g., [37, 39]
		f	All of the above
		g	No support; e.g., [6, 74, 75, 36, 41]
		h	Information not available; e.g., [27, 40]
IU	Incremental Updates	a	The tool supports incremental updates; e.g., [1, 3, 6, 18, 24, 76, 75, 28, 38, 41, 60, 55]
		b	No support; e.g., [9, 74, 29, 33, 35, 39, 69, 48]
		c	Information not available; e.g., [20, 51]

techniques. Considering everything is a model, allows to reuse transformations as HOTs. Henshin, MOLA, VIATRA, and Kermeta2 support HOTs transformations. For example a Kermeta program that write the Kermeta code for a model merge optimized for a given meta-model. ModelAnt provides an OO wrapper around the model, which provides reuse of the methods, while changing only the templates for a specific generation.

Incremental/Persistent Updates (IU): This facet refers to the way in which the model is updated, based on propagating the changes in the source model to the target model and vice versa. This feature increases the efficiency of transformations with large models, so that instead of regenerating a complete model, only the changed parts are recomputed to maintain consistency. On the contrary, there is non-incremental/statefull transitions, where the complete model must be regenerated. In-place transformations support incremental updates, TGGs also allow to specify incremental updates. Tools such as JTL, ModelAnt, Modelio, Umple, Actifsource, Acceleo, Fujaba, VMTS, EMorF, AToMPM, and MoTE are incremental, so that only the elements that need to be changed for consistency are updated. In Tefkat, if the trace and/or target models are partially populated, it can support incremental transformations. In Xpand, there is no possibility to transfer the model incremental [152]. In fact, it transfers a model from an abstract syntax graph to a textual representation, and the rules defined for an abstract syntax graph cannot be applied to text fragments.

5.6 Quality Category

This category is related to the quality aspects of the tool (Table 8).

Maturity of Tool (MT): This facet refers to using the existing tools exclusively in academic, industry or both of them.

Maintenance Support (MS): This facet refers to the life-cycle maintenance and support phase, including forums (e.g., TXL), mailing lists (e.g., Tefkat), bug tracker (e.g., GrGen.NET), or so on. Users would appreciate it if they know that their problems will be solved.

Concurrent Transformations (CT): The *Concurrent Transformations* facet concerns running two or more separate transformations by the same user at the same time. It is an useful feature for high performance transformations in large scale projects. Notice here that some features, such as orchestration can prevent to write on the same model elements at the same time, if the transformations share the same model in the memory. This facet supports in Xtend, MetaEdit+, Umple, MDWorkbench, MagicDraw, Together, AGG, BOTL, Henshin, and Blu Age. Previous version of VIATRA supports concurrent transformations, but its new version is single threaded due to Eclipse restrictions.

Live/active Transformations (LT): This facet indicates running model transformations in the background as daemons triggered by changes in the underlying models [146].

Model Comparison (MC): This facet shows that the tool supports model comparison in terms of comparing and detecting any similarities or differences between model elements. The output of model comparison can be used for human understanding, evolution management, or model versioning. Comparison tools can use heuristics, identity-based or signature-based approaches [133]. ModelAnt provides models comparison, and expresses the model differences in UML terms, such as added classes, and removed attributes. EA also presents the results of comparisons in a branch tree form. GroundTram supports simple node/edge level comparison and binary bi-similarity comparison. Some tools, such as MOMoT and VIATRA do not provide built-in model comparison, and use standard Eclipse plugins (e.g., EMF-compare) to provide Ecore model comparisons.

When/Where to Apply Transformations (WA): This facet indicates that tools can offer mechanisms for determining which model transformations can be appropriately applied in a given context [82]. Generally, it depends on the developer to decide where is the most effective place to implement one or another transformation. However, some beginners and non-expert users may find learning a model transformation language/tool a difficult task, so this feature can help them to work with the tool easily.

Level of Automation (LA): Considering the entire process of model transformation from a specification document down to an analysis model or code generation. These steps can be either completely automated, done manually, or use a certain amount of user intervention (semi-automated).

Traceability (TR): To have access to trace information in a model transformation, a record of links between the source and the target model elements is created. Traceability can be useful in traceing back to the origin of errors, performing impact analysis of the created/modified elements, determining which rule produced what, determining the source/target of a transformation in models synchronization, or checking the consistency of the models upon changes applied to any of them. Tools such as Tefkat, JTL, Modelio,

Table 8: Quality facets

Abb.	Facet	Attr.	Description
MT	Maturity of Tool	a	The tool has been used in academic; e.g., [4, 12, 20, 22, 75, 35, 46]
		b	The tool has been used in industry; e.g., [13, 25]
		c	The tool has been used in both academic and industrial world; e.g., [65, 24, 71, 33, 36, 37, 60, 48, 50]
		d	Information not available; e.g., [40]
MS	Maintenance Support	a	The tool provides complete support; e.g., [14, 11, 17, 18, 19, 24, 71, 76, 36, 60, 55, 25]
		b	The tool provides limited-support; e.g., [16, 20, 29, 67]
		c	No support; e.g., [15, 22, 73, 74, 41]
		d	Information not available; e.g., [40, 69]
CT	Concurrent Transformations	a	The tool provides concurrent transformations; e.g., [14, 15, 19, 76, 28, 33, 25]
		b	No support; e.g., [6, 65, 17, 20, 73, 74, 38, 41, 49]
		c	Information not available
LT	Live/active Transformations	a	The tool provides live/active Transformations; e.g., [76]
		b	No support; e.g., [6, 15, 17, 18, 22, 71, 75, 36, 39, 60]
		c	Information not available; e.g., [40]
MC	Model Comparison	a	The tool compares homogeneous models; e.g., [13, 18, 25]
		b	The tool compares heterogeneous models
		c	Results are in visual/model forms (e.g., UML); e.g., [13, 24, 25]
		d	Results are in textual forms; e.g., [18]
		e	All of the above
		f	No support; e.g., [9, 14, 15, 16, 17, 20, 71, 74, 75, 28, 33, 36, 39, 60, 67]
		g	Information not available
WA	When/where to Apply Transf.	a	The tool supports; e.g., [33, 69]
		b	No support; e.g., [5, 65, 11, 73, 76, 75, 28, 35, 38, 41, 67]
		c	Information not available
LA	Level of Automation	a	Manually
		b	Semi-automatic; e.g., [9, 16, 18, 20, 73, 75, 37, 39]
		c	Automatic; e.g., [11, 15, 17, 35, 36, 25]
		d	Information not available
TR	Traceability	a	Automatic (the transformation engine implicitly establishes the traceability information); e.g., [11, 15, 18, 19, 73, 35, 36, 55]
		b	User-defined (the user has to define the tracing links); e.g., [1, 12, 14, 16, 20, 71, 29, 37, 39, 41, 60, 67]
		c	Both automatic and user-defined traceability (traces are automatically generated if the user does not create any); e.g., [2, 17, 38]
		d	No support; e.g., [6, 65, 74]
		e	Information not available; e.g., [69]
IN	Interoperability	a	VCS (to control concurrent development of the source code by multiple developers)
		b	The tool provides with automatic import/export mechanisms for meta-models/models developed with other tools; e.g., [1, 3, 75, 35, 37, 46]
		c	Both VCS and import/export mechanisms; e.g., [14, 16, 17, 18, 19, 24, 71, 33, 69, 60]
		d	No support; e.g., [4]
		e	Information not available
AR	Automatic Report	a	The tool generates report/documentation; e.g., [13, 11, 16, 17, 18, 19, 24, 71, 60, 55, 49, 25]
		b	No support; e.g., [5, 6, 75, 28, 35, 36, 38, 69, 48]
		c	Information not available
SEC	Security	a	Obfuscate (to delete sensitive information from a confidential model); e.g., [11, 71]
		b	Read-only/Locked models; e.g., [6, 15, 17, 60, 49, 50]
		c	Code blocks; e.g., [71, 55]
		d	All of the above; e.g., [25]
		e	No support; e.g., [5, 9, 13, 14, 75, 28, 33, 38, 39, 46, 48]
		f	Information not available; e.g., [76, 69]

Table 9: Model transformation tools attributes

Col.1	Col.2 (Table 3)											Col.3 (Table 4)						Col.4 (Table 5)							
	General											Model-level						Transformation							
Tool	UP	OS	TP	A	AR	EU	EM	DA	E	ED	CS	ML	DD	LA	EXM	MH	MML	T	L	D	S	C	RS	RO	RAC
UML-RSDS	a	ab	e	a	abef	c	cd	a	b	c	ag	a	cd	c	b	d	bc	c	b	bc	d	e	cm	ab	a
Tefkat	c	d	e	a	g	bd	ad	a	b	b	abf	h	bd	d	a	a	d	a	c	c	abcdef	e	bkm	ac	c
JTL	a	d	d	a	bef	d	a	a	a	a	afg	h	cg	b	a	a	d	c	c	bc	cdef	e	bde	a	b
PTL	b	d	d	b	bef	d	ad	a	b	b	afg	a	bg	b	a	a	b	c	c	c	cd	e	bdi	a	a
ModTransf	c	d	e	a	abef	a	ad	d	a	b	agk	b	bd	d	a	b	bc	b	c	c	dg	ad	cdijm	ab	a
Echo	b	d	e	a	abef	c	ad	a	b	b	adg	a	bgj	b	a	a	b	a	c	bc	cf	a	e	e	b
MOMENT	d	e	e	e	abe	c	a	a	c	b	adg	a	bg	d	b	a	e	d	d	e	j	f	p	f	g
QVTR-XSLT	d	d	e	b	bef	c	b	a	c	b	adg	a	cd	d	b	e	e	d	d	c	j	a	p	f	a
ModelMorf	c	d	e	bc	abf	c	cd	a	b	b	adg	a	bi	b	a	e	d	c	c	d	cdef	ad	cikm	d	a
mediniQVT	d	d	e	bc	g	c	ad	a	c	b	adg	a	bg	d	a	a	e	d	d	bc	j	f	p	f	g
PETE	c	d	b	b	bef	d	a	a	b	b	ag	ab	bd	c	a	a	d	b	b	bc	cde	a	bdim	a	a
TXL	a	d	d	b	g	a	ad	a	b	a	a	ae	bi	b	a	d	d	c	c	c	i	e	cdik	c	a
ModelAnt	b	d	c	a	abef	b	d	a	a	b	aik	b	d	c	b	b	bc	c	c	c	degh	ab	a	f	a
Xtend	a	d	b	a	g	a	abd	a	a	b	m	a	bg	c	b	e	ab	c	c	c	cdfg	e	adgij	d	a
MetaEdit+	b	d	e	cd	g	a	abd	a	a	a	a	h	ai	c	b	e	d	c	c	c	i	e	ah	f	e
QVTo-Eclipse	a	d	b	a	bcdf	a	a	a	a	b	cg	i	bg	c	a	a	d	c	c	c	abcdef	e	adm	d	a
Kermeta2	c	d	b	a	g	a	ad	a	a	b	abg	af	bgj	c	b	a	d	c	c	c	i	e	aij	ab	a
Modelio	a	d	b	acd	g	ab	abd	cd	a	a	ag	adf	g	c	b	a	d	c	c	c	i	e	o	e	f
Umple	a	d	b	a	g	a	ad	a	a	a	ag	a	ci	c	b	e	bc	c	c	d	cdefgh	abd	af	b	a
MDWorkbench	b	d	b	bcd	abef	a	a	a	a	b	ag	i	bg	c	b	a	d	c	c	c	i	e	adj	b	a
Melange	a	d	b	a	abef	a	a	a	c	b	ag	h	bgj	c	b	a	d	c	c	c	i	e	adij	a	a
MagicDraw	a	d	b	cd	g	a	abd	a	a	a	acg	abdf	ad	c	b	a	d	c	c	c	i	e	ad	f	a
JAMDA	c	d	b	a	abef	a	cd	b	a	b	agk	b	k	d	b	e	e	a	c	c	j	ad	ad	ab	a
Ente. Arch.	a	d	d	bc	acdef	ab	ab	a	a	b	ag	adf	g	c	b	a	d	c	c	c	cdefgh	abd	p	f	g
OpenCanarias	d	d	e	e	aef	a	a	e	c	b	acg	a	cg	d	b	a	e	d	d	c	j	f	p	f	g
SmartQVT	c	d	e	a	cdf	a	a	a	a	b	acg	a	bg	d	b	a	e	d	d	c	j	f	aim	ab	a
SiTra	d	d	e	a	aef	a	ab	e	c	b	m	i	bg	d	d	a	e	d	d	c	j	f	cdj	f	ad
WebRatio	a	d	b	bcd	abdef	f	a	be	a	b	ag	ad	a	c	b	a	d	d	a	c	bcdfg	ab	adf	a	ad
Mitra2	b	d	e	a	ef	a	a	a	b	b	ag	a	bg	c	a	a	bc	c	c	c	cdf	e	cdgj	a	ad
JQVT	c	d	d	a	f	a	a	e	b	b	f	a	g	b	b	d	b	a	b	c	cf	ab	adi	b	a
Together	a	d	b	cd	abcd	ab	a	e	a	b	aceg	abde	bd	c	b	a	d	c	d	c	i	e	p	f	g
Merlin	c	d	e	a	abef	a	a	a	a	b	afg	a	cg	b	b	a	bc	a	c	c	cdfg	abd	adi	e	a
MOFScript	c	d	e	a	ade	ab	ad	e	c	b	agi	i	bg	d	b	a	e	d	c	c	j	f	adj	f	a
GROOVE	b	d	b	a	abef	c	cd	a	a	a	a	i	cg	a	a	d	b	b	b	c	c	a	n	ac	e
UMLX	d	d	e	e	ae	f	a	e	c	b	afg	i	cg	d	d	e	e	d	d	c	j	f	p	f	g
AToM3	c	d	e	a	abef	c	cd	a	c	b	gh	ag	ai	d	d	e	e	d	d	c	j	a	befgk	f	bd
AToMPM	b	d	e	a	abef	b	cd	a	a	b	ghj	ag	ai	c	c	d	d	c	c	c	i	a	aegj	e	bd
AGG	a	d	e	a	abef	c	cd	a	a	a	ah	ab	ai	c	a	d	b	b	c	c	i	e	begk	bc	cd
BOTL	c	d	e	a	ae	c	cd	e	b	b	af	a	ad	b	b	d	b	b	b	bc	d	a	bek	e	b
GRoundTram	b	d	e	a	abef	ac	cd	a	c	a	g	a	ch	d	a	d	b	b	c	bc	cdf	a	adi	c	b
eMoflon	a	a	e	a	abcef	c	a	a	a	b	a	a	cg	c	c	ae	b	c	c	bc	cdefgh	e	ahjk	ab	e
Henshin	a	d	e	a	abcef	c	a	a	a	b	agh	a	ag	c	c	a	b	c	c	c	cdfg	e	adejk	a	bd
MoTE	a	d	b	a	abef	c	ad	a	b	b	ag	h	ag	c	c	a	d	c	b	bc	i	a	bek	c	a
GReAT	b	a	e	b	abcef	c	b	e	a	b	m	i	aij	d	d	e	e	d	d	c	j	f	adeik	ac	b
TGGInterpreter	d	d	e	b	abef	c	a	a	a	b	ag	h	ag	c	a	a	d	a	c	d	abcdef	e	bej	ab	c
MOMoT	b	d	c	a	bef	ab	a	a	c	b	a	i	cg	d	c	a	d	c	c	c	abcdef	a	be	ac	c
EMorF	d	d	e	a	abef	ac	a	e	a	b	ag	i	ag	d	a	a	e	c	d	bc	j	a	p	f	g
PROGRES	d	b	e	b	acef	c	cd	a	c	c	m	i	k	d	c	e	e	d	d	c	j	a	aeij	f	bc
MoTMoT	d	d	e	a	abef	c	d	e	a	b	agk	i	ai	d	b	b	e	d	d	c	j	f	adj	b	a
UMT	c	d	e	a	abef	a	cd	e	c	c	m	i	b	d	b	e	e	d	d	c	j	a	p	f	g
VIATRA	b	d	e	a	g	c	a	a	a	b	ag	acdefg	bg	c	c	a	d	c	c	c	i	e	cdefl	ac	bcd
Eclectic	b	d	e	a	bef	ab	a	a	a	c	a	a	bg	d	b	ae	d	c	c	c	abcdefg	e	cd	d	a
Epsilon	b	d	b	a	g	ab	ad	a	a	c	agj	i	bg	c	d	c	e	c	c	e	j	f	cij	ab	d
AGE	c	d	e	a	abef	a	ad	a	a	c	ag	a	bg	d	a	a	d	c	c	c	abcdefg	e	bdim	ab	a
VMTS	b	a	a	b	abef	abc	bd	a	a	b	cdeg	aef	ci	c	c	e	d	c	c	c	i	b	adgl	ab	ad
ATL	b	d	e	a	g	a	a	a	a	c	afg	a	bg	d	b	c	d	c	c	c	i	e	ceim	ab	a
Fujaba	b	d	e	a	abef	ac	ad	a	a	a	a	a	ad	c	b	e	d	c	c	c	i	a	adfij	d	abc
GrGen.NET	b	d	a	a	g	ac	cd	a	a	b	a	i	bgj	d	c	e	d	c	c	c	abcdefg	a	aefglm	ac	e
Rational	d	d	e	e	g	a	abd	e	c	c	m	i	k	c	d	e	e	d	c	e	j	f	p	f	g
Blu Age	a	a	c	c	abce	abc	a	a	a	b	afghij	ae	cg	c	b	a	d	c	c	d	i	e	n	d	e
MOLA	b	a	d	b	abef	c	a	a	b	b	ag	ad	ai	c	b	ae	b	c	b	c	i	a	adj	ac	ab
Acceleo	b	d	b	a	g	ab	abd	a	a	c	agik	abdfg	bg	c	b	c	b	a	a	c	degh	ab	adj	ab	a
AndroMDA	b	d	c	a	g	a	ad	e	a	b	agk	i	bd	c	d	c	e	d	d	c	j	f	p	f	ad
Xpand	c	d	b	a	g	a	a	e	a	b	ag	i	bg	d	a	a	e	d	d	c	j	a	cdgijm	d	a
Actifsource	a	d	e	bcd	g	a	a	e	a	b	a	a	cg	c	b	a	bc	a	a	c	dg	e	ad	f	ad

Table 10: Model Transformation Tools attributes2

Col.1	Col.2 (Table 6)				Col.3 (Table 7)										Col.4 (Table 8)											
	Capability				Implement										Quality											
Tool	V	VA	I	O	ET	MP	RE	RT	EDI	WPM	TS	SYE	SE	RUT	IU	MT	MS	CT	LT	MC	WA	LA	TR	IN	AR	SEC
UML-RSDS	abdeh	d	a	ab	e	b	b	b	c	c	a	h	ab	a	a	a	c	b	b	f	b	b	b	b	b	e
Tefkat	j	a	a	b	e	b	b	b	a	a	d	ad	bdf	ac	b	a	c	b	b	f	b	c	c	b	b	e
JTL	ac	d	a	ab	abd	b	b	b	a	a	d	g	bd	g	a	a	c	b	b	f	b	b	c	b	b	e
PTL	ag	a	a	b	abd	b	b	b	a	a	d	g	bd	a	b	a	c	b	b	f	b	b	a	d	b	e
ModTransf	j	d	a	ad	abd	b	b	b	d	b	d	g	i	abc	b	a	c	b	b	f	b	a	d	b	b	e
Echo	acgh	b	a	b	e	b	b	c	c	a	d	f	bd	g	a	a	c	b	b	f	b	b	d	b	b	b
MOMENT	j	d	a	h	e	b	b	b	a	a	d	h	j	h	b	a	c	b	b	g	b	b	a	b	b	f
QVTR-XSLT	j	d	a	abf	e	b	b	b	a	a	d	h	j	h	b	a	c	b	b	f	b	b	a	b	b	e
ModelMorf	c	d	a	ab	abd	b	b	b	b	b	d	g	b	a	b	a	c	b	b	f	b	b	a	b	b	e
mediniQVT	k	e	a	h	e	b	b	b	d	a	e	acd	bcd	h	a	c	d	b	b	f	b	b	a	b	c	f
PETE	j	d	a	b	e	b	b	b	a	a	d	g	bd	ac	b	a	c	b	b	f	b	b	b	b	b	e
TXL	j	d	a	acdef	e	b	b	b	b	b	d	abcd	bdg	ac	b	c	a	b	b	f	b	b	d	d	b	e
ModelAnt	j	d	a	g	e	b	a	b	a	a	d	g	d	h	a	b	b	b	b	ac	b	b	d	b	a	e
Xtend	j	c	a	g	e	b	b	b	c	a	c	f	h	abd	a	c	a	a	b	f	b	b	b	c	b	e
MetaEdit+	c	b	c	bcd	e	a	a	a	c	a	c	abd	abd	abcd	a	c	a	a	b	abd	b	c	a	c	a	a
QVTo-Eclipse	j	d	c	ab	e	b	b	b	a	a	b	acde	bdeg	ab	c	d	c	a	b	f	b	c	a	b	b	b
Kermeta2	c	a	c	abcf	e	a	b	b	a	a	c	f	abdf	f	b	a	b	b	b	f	b	b	b	c	a	e
Modelio	c	a	a	abcde	e	b	a	a	a	a	c	f	h	g	a	c	a	b	b	f	b	c	c	c	a	b
Umple	acg	c	a	abcde	e	b	a	b	c	a	c	ae	bd	a	a	c	a	a	b	ad	b	b	a	c	a	e
MDWorkbench	j	a	a	abce	e	b	a	b	a	a	c	f	bd	g	b	c	a	a	b	f	b	b	a	c	a	a
Melange	c	d	a	abc	e	a	b	b	a	a	e	f	abcdef	acd	c	a	b	b	b	f	b	b	b	b	b	d
MagicDraw	acd	c	c	abcdf	e	b	a	a	a	a	c	f	abd	acd	a	c	a	a	b	f	b	b	a	c	a	b
JAMDA	j	d	a	bd	abd	b	b	b	b	b	d	g	j	h	b	a	c	b	b	f	b	a	e	b	b	e
Ente. Arch.	j	c	a	abde	e	b	a	a	c	a	c	f	bcde	g	a	c	a	b	b	ac	b	b	a	c	a	a
OpenCanarias	j	d	a	h	e	b	b	b	d	c	d	h	j	h	c	d	d	b	b	f	b	b	a	e	c	f
SmartQVT	j	d	a	h	e	b	b	b	a	a	d	h	j	h	b	a	c	b	b	f	b	b	a	b	b	e
SiTra	j	d	a	h	abd	b	b	b	d	c	d	h	j	h	b	a	c	b	b	f	b	a	a	e	b	e
WebRatio	g	a	a	cd	e	b	a	b	a	a	c	f	bdf	a	b	c	a	b	b	f	b	b	b	c	a	ac
Mitra2	j	d	a	ab	e	b	b	b	a	a	d	acde	bcde	a	b	a	c	b	b	f	b	b	a	b	b	e
JQVT	j	d	a	b	abd	b	b	b	a	a	d	ac	bc	g	b	a	c	b	b	f	b	b	d	b	b	e
Together	k	c	c	ab	e	b	a	a	c	a	d	f	h	abcd	a	c	a	a	a	acd	b	b	a	c	a	f
Merlin	j	d	a	abd	e	b	b	b	a	a	d	acd	bcd	g	a	a	c	b	b	f	b	b	a	b	b	e
MOFScript	k	e	a	abcd	e	b	b	b	d	a	e	h	j	h	b	a	c	b	b	f	b	b	a	e	b	f
GROOVE	abeg	b	a	a	e	b	b	b	c	a	d	ac	bd	g	b	a	b	b	b	f	b	b	b	b	b	e
UMLX	j	d	a	h	abd	b	b	b	d	a	d	h	j	h	b	a	c	b	b	f	b	b	e	e	b	b
AToM3	k	e	a	h	e	a	b	b	d	a	e	g	j	h	c	a	c	b	b	f	b	b	b	e	b	e
AToMPM	j	b	a	abf	e	a	b	b	a	a	c	g	bd	ab	a	a	b	a	b	f	b	b	b	a	b	e
AGG	abceg	b	a	ab	e	b	b	b	c	a	d	g	abd	a	b	c	a	a	b	f	a	b	b	c	b	e
BOTL	abcde	b	a	b	e	b	b	b	a	c	d	g	i	b	b	a	c	a	b	f	b	b	d	b	b	e
GRoundTram	bce	d	a	abf	e	b	b	a	c	a	d	a	bd	g	b	a	b	b	b	f	b	c	a	b	b	e
eMoflon	ac	d	a	abc	e	b	b	b	c	a	a	ac	bcd	g	a	c	a	b	b	f	b	c	a	b	b	e
Henshin	abc	a	a	abd	e	b	b	b	a	a	d	h	abd	ae	b	c	b	a	b	f	b	b	b	b	b	e
MoTE	abcdeh	a	a	b	e	b	b	a	a	a	d	g	b	g	a	a	b	a	b	f	b	c	c	b	b	e
GReAT	k	e	a	h	e	b	b	b	d	a	e	h	j	h	c	d	d	c	c	f	b	b	e	e	c	f
TGGInterpreter	ac	d	a	b	e	b	b	b	a	a	d	g	bd	g	a	a	c	b	b	f	b	b	b	b	b	e
MOMoT	j	d	a	abc	e	b	b	b	a	b	d	f	abce	ab	b	a	d	b	b	f	a	b	e	c	b	f
EMorF	k	a	a	h	f	b	b	b	d	a	e	h	j	h	a	d	d	b	b	f	b	b	a	e	b	f
PROGRES	k	e	a	h	f	b	b	b	d	c	e	h	j	h	a	a	c	b	b	f	b	b	e	e	b	f
MoTMoT	k	e	a	h	abd	b	b	b	d	c	d	h	j	h	c	d	c	b	b	f	b	b	a	e	b	e
UMT	k	d	a	h	e	b	a	b	c	a	d	h	j	h	b	a	c	b	b	f	b	b	d	b	b	e
VIATRA	abcdgh	a	a	g	e	a	b	b	c	a	d	f	bcdfg	f	a	c	a	b	a	f	a	c	c	c	b	ab
Eclectic	j	d	a	b	e	b	b	b	b	a	c	abcd	bef	abc	b	a	c	b	b	f	b	b	b	b	b	e
Epsilon	k	a	a	h	e	b	c	c	d	a	e	h	j	h	c	c	d	c	c	abc	b	b	e	c	c	f
AGE	j	d	a	bd	abd	b	b	b	c	a	c	abce	bf	h	b	c	c	b	b	f	b	b	b	b	b	e
VMTS	ab	b	a	abcdf	e	a	b	b	c	a	b	abc	bd	ac	a	c	a	b	b	f	b	b	b	b	a	b
ATL	j	d	a	abf	e	b	b	b	a	a	e	f	h	h	b	c	a	b	b	f	b	b	b	c	b	b
Fujaba	acf	c	a	abd	e	b	a	a	a	a	c	abcd	bcdefg	abcd	a	a	c	b	b	f	b	b	b	c	b	e
GrGen.NET	f	c	c	abcf	e	b	b	b	b	b	d	a	bdg	abcd	b	c	b	a	b	f	b	b	b	c	b	a
Rational	k	e	c	h	e	b	a	a	d	a	e	h	j	h	c	c	a	c	c	g	c	d	e	c	a	f
Blu Age	cg	a	c	g	e	b	a	a	c	a	c	f	h	abcd	a	b	a	a	a	ac	a	c	a	c	a	d
MOLA	j	a	a	ab	e	b	b	b	a	a	d	c	abd	ae	b	a	b	b	b	f	b	b	b	b	b	e
Acceleo	j	d	a	cdf	e	b	a	b	a	a	d	f	abcdeg	a	a	c	a	b	b	f	b	b	a	b	a	c
AndroMDA	k	e	a	h	e	b	b	b	d	c	e	h	j	h	a	d	d	b	b	f	b	b	e	e	c	f
Xpand	k	e	a	h	e	b	b	b	a	a	d	h	j	h	b	d	b	b	b	f	b	b	b	c	b	f
Actifsource	a	a	a	cd	e	b	b	b	a	a	c	f	h	g	a	c	a	b	b	f	b	b	b	c	a	b

Table 11: Evaluation of model transformation tools

Tool	Large and Complex Transformation		Performance		
	Handling	Quality	Memory Usage	Time	Disk Usage
UML-RSDS	Medium	Medium	-	~1s	~60MB
Tefkat	Medium	Low	~512MB	~>2s	-
JTL	Low	Low	~512MB	~>2s	~70MB
PTL	Low	Medium	~512MB	~>3s	-
ModTransf	Low	Low	~512MB	~>3s	~70MB
Echo	Low	Low	~512MB	~>2s	70MB
MOMENT	Low	Low	~512MB	-	-
QVTR-XSLT	Low	Low	-	-	-
ModelMorf	Low	Low	~100MB	~>3s	~50MB
mediniQVT	Medium	Low	~512MB	-	-
PETE	Low	Low	-	~>3s	-
TXL	Medium	High	1G	~>3s	50MB
ModelAnt	Medium	High	512MB	~2s	100MB
Xtend	High	High	512MB	~1s	~100MB
MetaEdit+	High	High	~512MB	~1s	~90MB
QVTo-Eclipse	Medium	Medium	~512MB	~>2s	-
Kermeta2	Medium	Medium	~512MB	<~2s	-
Modelio	High	Medium	1GB	~<3s	90MB
Umple	Medium	Medium	~1GB	~1s	20MB
MDWorkbench	Medium	High	512MB	~1s	30MB
Melange	Low	Low	512MB	~2s	100MB
MagicDraw	High	High	1GB	~1s	-
JAMDA	Low	Low	256MB	~2s	100MB
Ente. Arch.	High	High	256MB	<~1s	50MB
OpenCanarias	Medium	Low	~512MB	<-	-
SmartQVT	Medium	Low	~512MB	~2s	-
SiTra	Low	Low	~512MB	-	-
WebRatio	Medium	High	~512MB	<-	-
Mitra2	Medium	Medium	~512MB	>~2s	~50MB
JQVT	Low	Low	~512MB	~2s	~50MB
Together	High	Medium	~1G	1<~1s	~50MB
Merlin	Medium	Low	~512MB	~2s	~70MB
MOFScript	Low	Low	~512MB	-	-
GROOVE	Medium	Medium	~256MB	~2	30MB
UMLX	Low	Low	~512MB	>-	-
AToM3	Low	Low	2GB	>-	250MB
AToMPM	Low	Low	2GB	~2s	250MB
AGG	Medium	Low	-	~>2s	-
BOTL	Medium	Low	-	~2s	-
GRoundTram	Medium	Low	-	~2s	-
eMoflon	Medium	Medium	~512MB	<~1s	~70MB
Henshin	Medium	High	~200MB	~1s	~60MB
MoTE	Medium	Medium	~512MB	~2s	~100MB
GReAT	Medium	-		-	
TGGInterpreter	Medium	Low	~512MB	~>2s	~70MB
MOMoT	Low	Low	~512MB	~2s	~70MB
EMorF	Medium	Low	~512MB	-	-
PROGRES	Low	Low	-	-	-
MoTMoT	Low	Low	-	-	-
UMT	Low	Low	-	>-	-
VIATRA2	High	High	~512MB	~1s	~50MB
Eclectic	Medium	Medium	~512MB	~2s	-
Epsilon	High	Medium	~512MB	>-	-
AGE	Medium	Low	~512MB	~>2s	-
VMTS	Medium	Medium	512MB	~1s	65MB
ATL	High	Medium	~512MB	<~1s	~50MB
Fujaba	Medium	High	~512MB	~1s	
GrGen.NET	Medium	High	1GB	~1s	60MB
Rational	-	-	-	-	-
Blu Age	High	Medium	2GB	~1s	100MB
MOLA	Medium	Medium	~512MB	<~1s	~50MB
Acceleo	Medium	Medium	256MB	~1s	~64MB
AndroMDA	Medium	Medium	256MB	-	-
Xpand	Low	Medium	~512MB	~>2s	~70MB
Actifsource	High	Medium	2GB	~1s	64MB

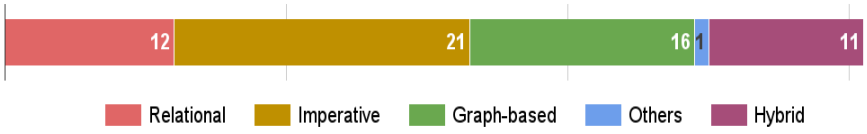
MoTE, and VIATRA support both automatic and user-defined traceability. MoTE automatically create and maintain a traceability model between the transformed models. It is used by the tool to check/maintain the consistency of the two models upon changes applied to any of them. Graph-based tools are suitable for performing in-place transformations, hence, they cannot be used to record traceability links between source and target graph model elements. Winkler et al. [144] survey several aspects of traceability research both in the requirements engineering and the modeling area.

Interoperability (IN): Generally, a MDD tool does not support all the required tasks of model transformation process, thus different tools must be able to work with each other. The *Interoperability* facet shows the ability of the tool to integrate with other tools to exchange models/information, and to use the exchanged models/information. Efficient importers/exporters make able the tool to cooperate with any external systems. Some tools do not provide built-in exporters in standard formats for other tools, for example, in ATOMPM models are saved in a tool-dependent JSON format, then user can export them.

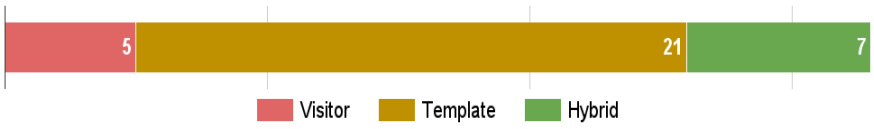
It is necessary to detect conflicting modifications, overlapping between concurrently modified versions of a software artifact, or merging of modifications. Version control systems (VCSs), e.g., MDWorkbench, EA, WebRatio, Together, can help to detect, and resolve these conflicts in order to obtain an uniformed version. In addition, using VCSs can result in the re-usability of the stored meta-models/models. Team members working on transformations, usually need standard VCSs for their transformations. VCSs technologies are based on lock-modify-unlock and copy-modify-merge approaches [134]. In the MDD-context, VCSs should provide both textual versioning and graphical representation.

Automatic Report/Documentation (AR): The *Automatic Report* facet indicates that the tool has a built-in reporting and documentation feature. For example, Kermeta2 can create a JavaDoc like documentation, and simple class diagram for the Kermeta program, or ModelAnt generates documentations in RTF and WIKI formats

Security (SEC): This facet can be used to limit user access to the models, or prevent some accidental changes/updates of the referenced models. Besides the obfuscation, read-only/locked models, and code blocks, there are other types of security, such as role-based (e.g., Modelio) that uses user rights to control the access to different parts of transformation, such as meta-model. It is also important that access to repositories or running a generator can be secured by passwords. In MetaEdit+, the generated code is obfuscated, and variables and function names are generally produced from the model text with a user-defined translator.



(a) Number of M2M tools



(b) Number of M2T tools

Figure 1: Number of model transformation tools

6 Evaluation of Tools

We estimate how well the various MDD tools may perform to handle large and complex transformations based on the feedback from the developers who contributed to these tools and users who frequently used these projects, the published papers, and online documentations. Thus, this is not an actual evaluation, it just provides an overall picture of the potential of each tool to work with large and complex models.

An optimal functionality of a transformation tool includes producing complete and correct results. Therefore, we also estimate the quality that can be achieved during transforming of large and complex models

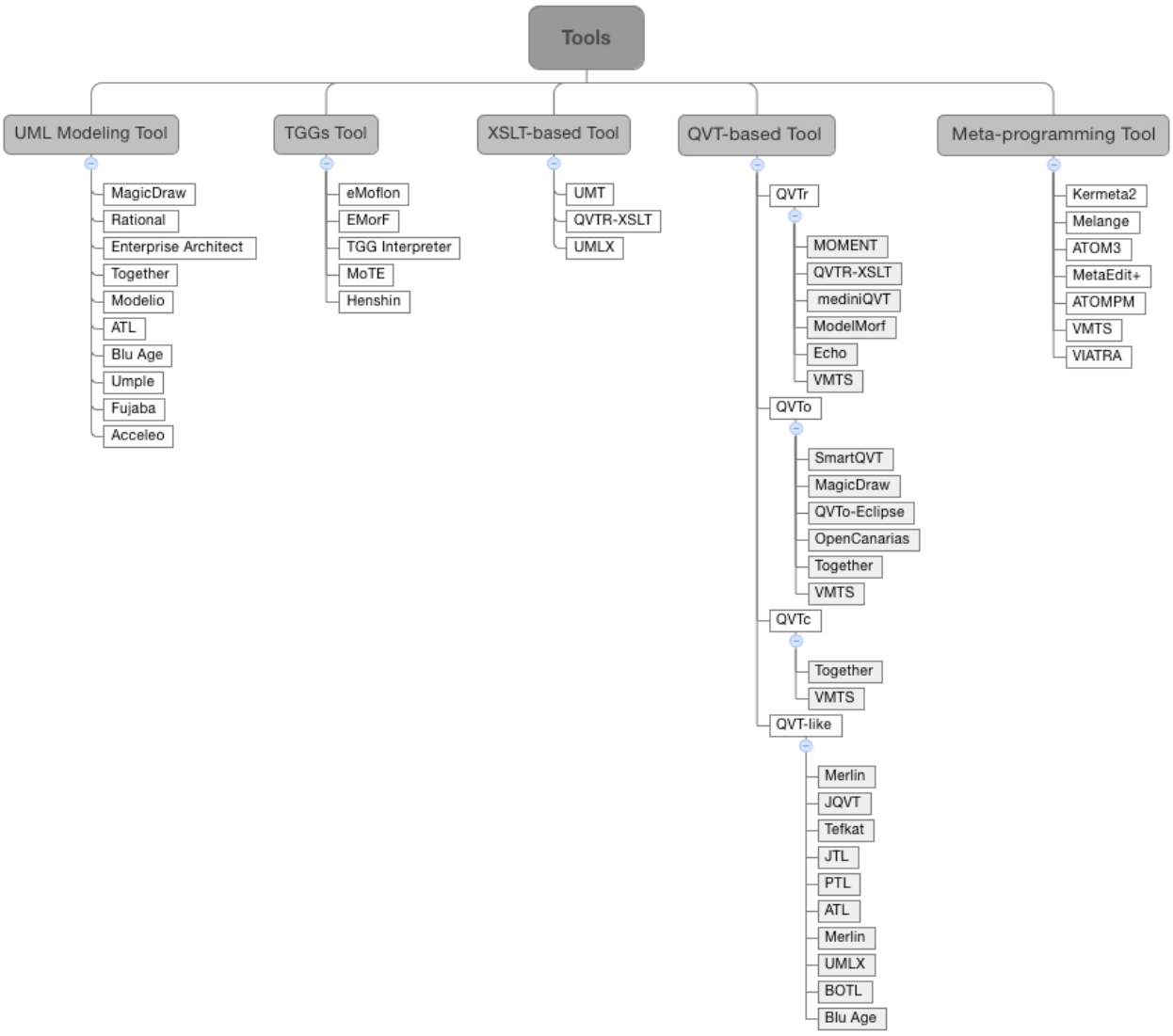


Figure 2: Categorizing the tools

without sacrificing validity, correctness, and performance. To measure the complexity of a transformation, metrics such as the number of the expressions, transformation rules (e.g., 80) and elements (e.g., few million for example 10^7), the number of calls, and the number of recursive calls can be considered. In Table 11, *low* means the capability of tool to handle complexity is not very efficient in comparison to *medium*. On the contrary, *high* shows the maximum capability to handle the complex models.

In the following, we take into consideration the appropriate requirements of a transformation tool to cope with the development of complex and large transformations:

- Type of deployment environment, in terms of using an IDE or being a standalone application. For instance, Eclipse-based tools are engaged with some unrelated tasks that result in slowing down the process of transformation,
- Having external dependencies to other tools may also increase the transformation time,
- Type of execution mode, as compiler or interpreter-based can affect the performance,
- Using the model repositories for handling large models,
- Using control-flow structures to control the execution order of rules,

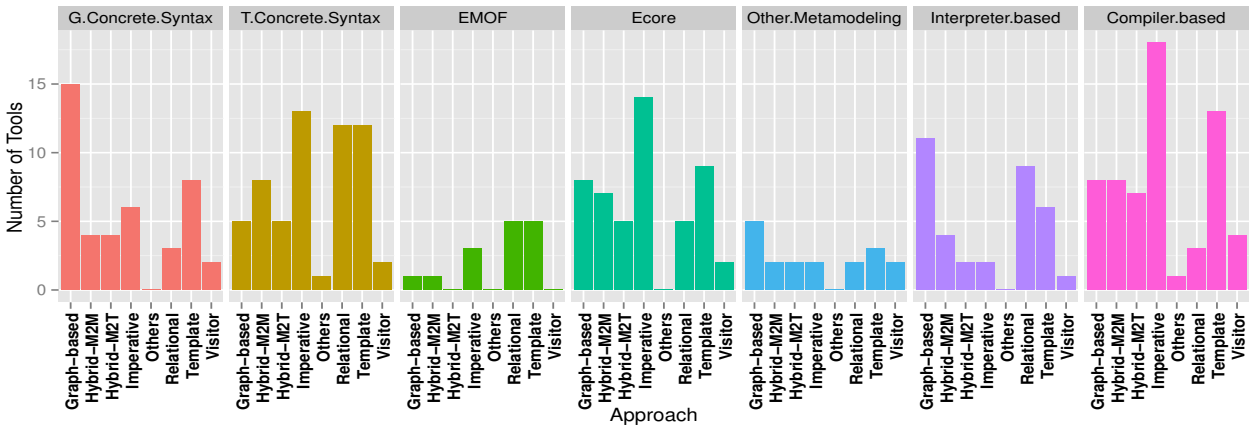


Figure 3: Comparison of the tools based on the Model-level facets

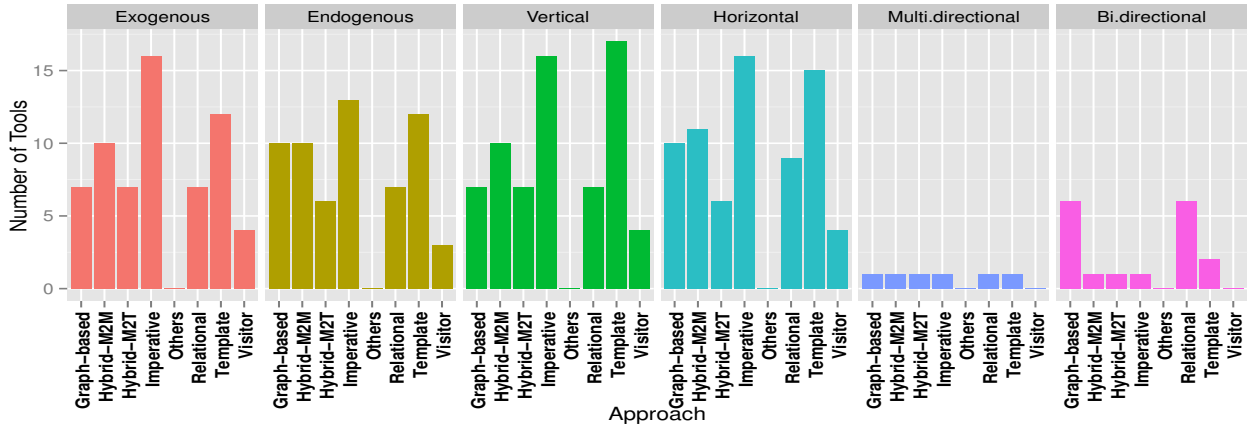
- Having the property of bi/multi-directionality can decrease the complexity of transformations through requiring to fewer transformation rules,
- Using re-usability and modularization techniques can increase the scalability of a transformation tool
- Supporting incremental feature that avoids the need to regenerate complete models when only some of the elements of source models are changed,
- Textual or graphical representations is another important factor to deal with large and complex models. In large and complex transformation scenarios, textual representation can be hard to read and maintain, meanwhile the visual representation can also be so complicated for complex transformation rules. Therefore, a hybrid of both textual or graphical approach can be an appropriate option,
- Supporting concurrent transformations,
- Completely automated tools are faster than semi-automatic and manually ones.

Generally, relational tools do not perform very well in transformation scenarios including, complex mappings, significant and sequential processing. When models grow in size and number, the performance of this approach drops obviously. Some of the relational-based tools, e.g., ModelMorf, can handle very large models using model repositories. However, tools executed by an interpreter, may not cope with large models very well.

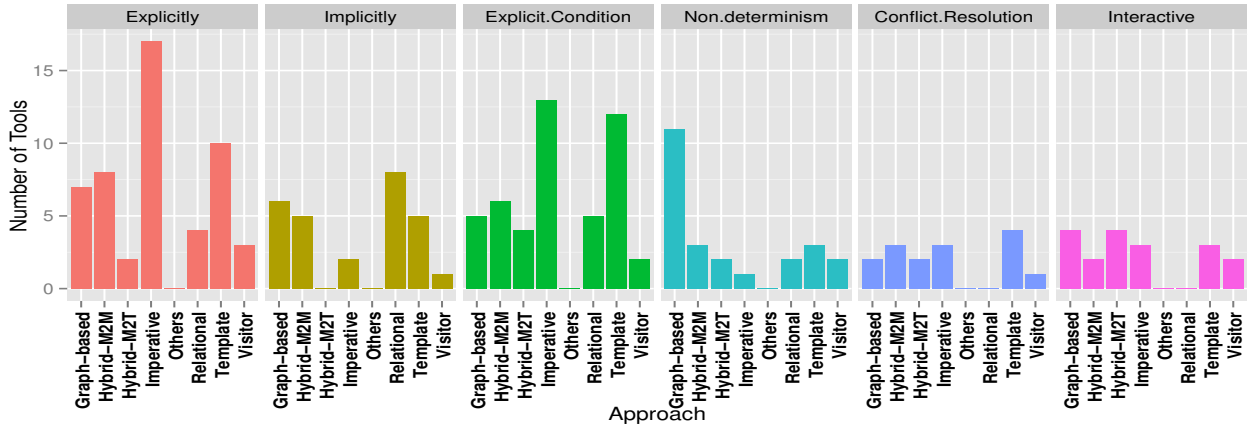
Imperative tools operate at a low-level of abstraction, so are capable of addressing different large and complex transformations scenarios. In Figure 3 the number of tools that are compiler-based outnumber the tools in others approach. Being interpreter-based can increase the overall execution time. Some authors/companies provide a benchmark to show the performance of their tools. For example, the developers of MetaEdit+ showed the scalability of the tool with a model repository of $5G$ and models of over 1 million model elements. Tefkat was tested with models of a few million objects, and up to 40 rules.

Graph-based tools can perform differently in handling large and complex transformations. In Henshin, modularity and nested rules can be used to specify complex transformations. PROGRES supports incremental update, but operates interactively, which causes the problems in handling large models. Some of graph-based tools also have scability limitations. In GROOVE, control and application strategy features can help to handle very complex transformations very well. However, large models (e.g., models with 10^5 nodes) do not affect validity and correctness, but slow down the tool. Expressing of very complex transformations with the TGGs formalism is almost difficult. Hence, handling of complex languages in most TGGs-based tools can be challenging [151].

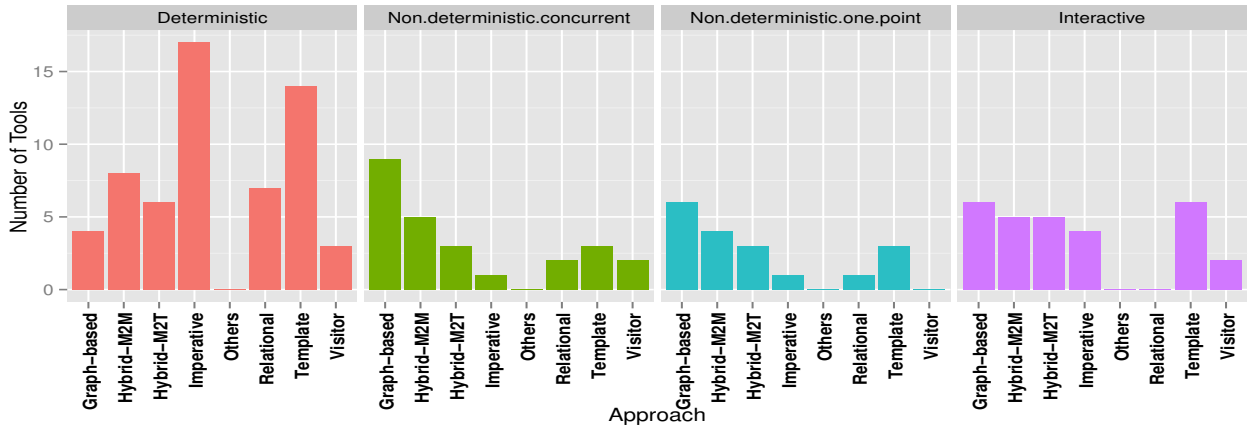
Hybrid languages usually combine a relational rule-based execution scheme with imperative features, thus they perform very well in handling complex transformation scenarios. A combining of the graph-pattern-matching with imperative, and OO languages increase the performance of hybrid tools. For example, GrGen.NET and VIATRA. Fujaba, can scale up for transforming very large models using the control-flow



(a) Transformation features



(b) Rule scheduling features



(c) Rule application control features

Figure 4: Comparison of the tools based on the Transformation facets

constructs such as sequence and loop, and an local-search based graph pattern matching algorithm. Incrementality feature in graph-based tools, e.g., MoTE, TGGInterpreter, can help to handle large models more efficiently, through avoiding regenerating the complete models. Another example is the hybrid nature of ATL that makes it able to express complex transformations.

Template-based tools perform better in coping with large and complex transformations than visitor-based ones. Hybrid tools, e.g., Actifsource, MetaEdit+, combine different concepts and features of visitor and template-based approaches, perform very well in handling complex transformations.

The performance of transformation is another important factor that need to take into account. Table 11 shows the performance result of a small transformation (e.g., 10^3) in terms of *execution time*, *memory* and *disk* usage on a standard PC workstation. Tools based on a compiler or run in a virtual machine (e.g., ATL) execute faster than other ones. On the other hand, compiler-based tools generate intermediate object code which needs more memory.

7 Discussion of Tool Facets

We now give an overview of the study results based on the assessing of 65 tools. Figure 1 shows the number of the tools in each category. It seems that imperative approaches are more attractive because of particular services such as traceability management and the explicit control over the transformation execution. Visual notation of graphs also makes it a popular approach for model transformations. The figure also shows that the majority of the M2T tools, 21, support template-based approach having a structure similar to the code to be generated.

Some of the assessed tools are not mainly designed for model transformations. Therefore, they may not support some of the features, whereas provide others not mentioned in this paper. For instance, Melange is not a transformation language, and transformations are expressed in a plain Xtend code. Melange is able to write generic model transformations that can be applied on models conforming to the different modeling languages, based on the model typing. GROOVE is mainly used for editing graphs and graph transformation rules, exploring and model checking the state spaces of graph grammars, so supports transformation as a complementary task. Henshin can also support state space analysis, and distributed/large-scale graph transformations via Apache Giraph. Modelio supports features, such as requirement analysis integrated in the model, world-wide modeling which means federated models interconnected through network (e.g. WEB), and impact analysis management. Eclectic supports native handling of Java objects [147]. VIATRA supports model transformation workflows and chains, and model synchronization through change-driven model transformations, which rely upon the history of model changes [132].

Figure 2 categorize the tools based on being a UML modeling, TGGs-based, XSLT-based, QVT-based,

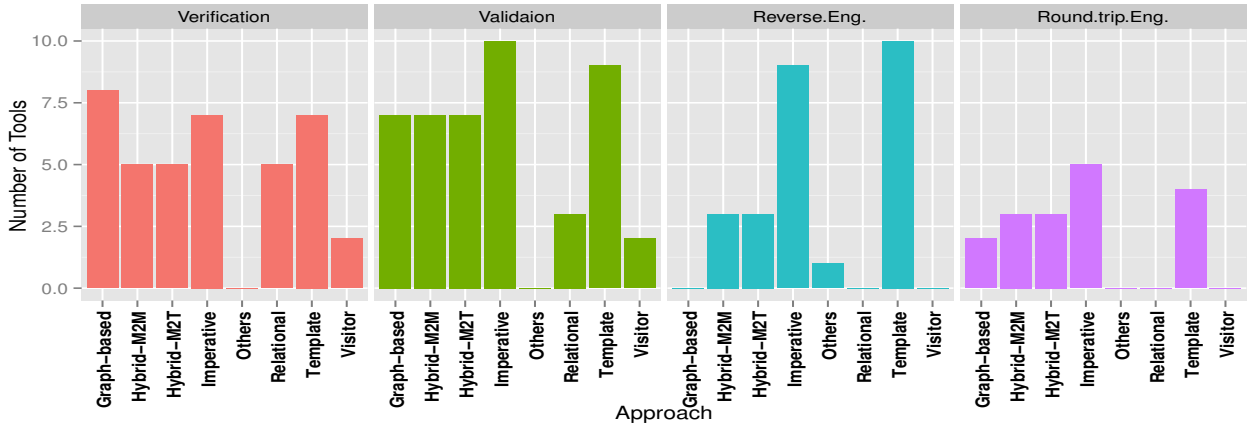


Figure 5: Comparison of the tools based on the Capability facets

or meta-programming tool. EA and Modelio are examples of UML modeling tools; TGG Interpreter and eMoflon are TGG-based; UMLX uses XSLT; Echo and ModelMorf are QVTr; MagicDraw and QVTo-Eclipse are QVTo-based; Together is QVTc, previous version of VMTS was QVT-based; Merlin, JQVT, JTL, PTL, ATL, BOTL are QVT-like; and Kermeta2, VIATRA, ATOMPM and Melange are meta-programming tools.

Figure 3 provides information related to the number of tools in each approach, which provide graphical/textual concrete syntax, employ EMOF/Ecore/other meta-modeling languages, and are compiler/ interpreter-based. Among M2M and M2T tools, the number of graph-based, 15, and template-based, 8, tools which

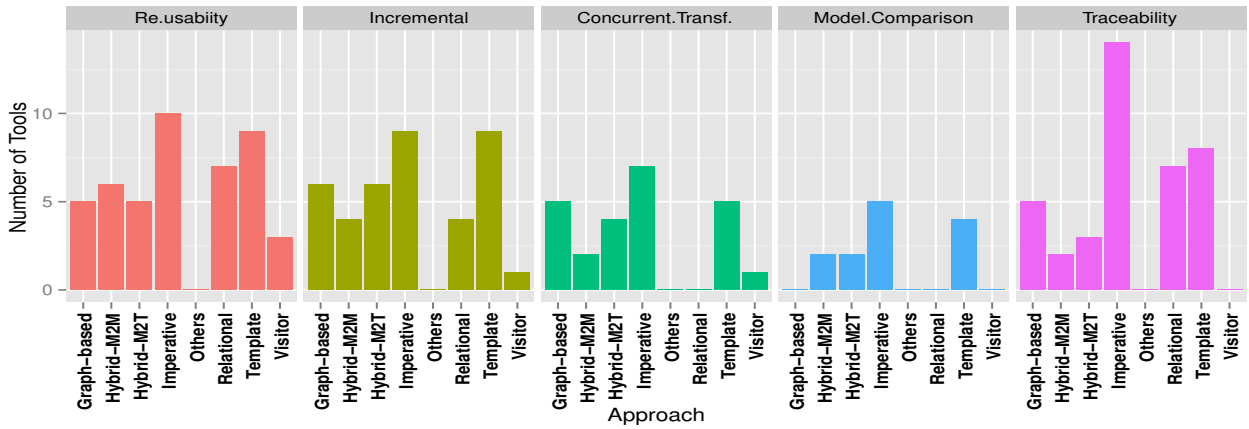


Figure 6: Comparison of the tools based on the Implementation and Quality facets

provide graphical concrete syntax outnumber the tools of the other approaches. 13, 12, and 13 tools of imperative, relational, and template-based can define textual concrete syntax respectively. Ecore is the popular meta-modeling language particularly among imperative tools, and the tools that use other meta-modeling are more than the tools based on the MOF. The figure also provides information on the number of tools in each approach that are based on the compiler or interpreter. The compilation-based is mostly used in imperative (18), and template-based (14) tools, whereas 11 of graph-based and 9 of relational tools are interpretation-based.

Figure 4-a compares the tools based on the *Transformation* facets. Considering the exogenous and endogenous transformations, a large number of imperative tools support the exogenous than endogenous transformations, while most graph-based tools support endogenous transformations than exogenous ones. The main observation is that vertical and horizontal are by far the most popular transformations in imperative and template-based tools. The figure also shows that 14 of tools support bi/multi-directional transformations. QVTr and QVTc can support multi-directional rules, while they are uni-directional in the Operational Mappings. Multi-directional rules can facilitate the traceability management. QVTr specification defines the conditions under which a transformation can be bi-directional, thus not all transformations are automatically bi-directional. QVTr-based Tools, e.g., ModelMorf, can support both uni-directional and bi-directional model transformations.

Figure 4-b gives information on the features of transformation rules used in the tools. 39 of tools support explicit form of rule scheduling in comparison to 22 tools which support implicit form. In this way, the form of rule scheduling in 17 of imperative, 11 of template-based, and 7 of graph-based tools is explicit. 32, and 17 of the tools support explicit condition and non-determinism rule scheduling respectively. In graph-based tools, graph constructs can randomize rule scheduling, so default strategy for selection rule is non-deterministic. In non-determinism, conflict rules can happen which causes unexpected results. AGG and Henshin uses a so-called critical pair analysis (CPA) technique [135] to detect conflicts between rules and show the termination criteria of graph transformations. Critical pairs reefers to rules with a common LHS which delete an element to be used by the other rules. If all critical pairs will be confluent, the complete transformation system is considered to be terminated. Furthermore, different rule execution sequences in non-determinism strategy can result in different results. However, there are graph-based tools, e.g., Henshin and GRoundTram, which support a determinism scheduling mechanism. Template approaches usually offer user-defined scheduling in the internal form of calling a template from within another one. In Acceleo, rules (or individual templates) are explicitly called by the name from another template, at the exception of those that start the generation. For a given name, several templates might apply, some with guard predicates or some applicable to more specific type, In that case, the engine performs the dispatch, through checking the hierarchy to find the most specific template which a guard evaluates to be true.

Figure 4-c also shows information related to the number of tools in each approach that support rule application control with different mechanisms, such as deterministic or interactive. The majority of imperative, 17, template-based, 14, and hybrid, 8, tools support deterministic application control, in comparison to graph-based, 15, tools with non-deterministic control. GROOVE provides a combination of rule application

control strategies to find all locations to which a rule can be applied. Based on an exploration strategy, it is possible to explore some of the rules (constructs the target graph/model). In that way, the user can choose to explore all, one deterministically (i.e., the same one every time the same transformation is run), one randomly, or one manually.

Figure 5 compares the tools based on the *Capability* facets. A MDD tool should support not only

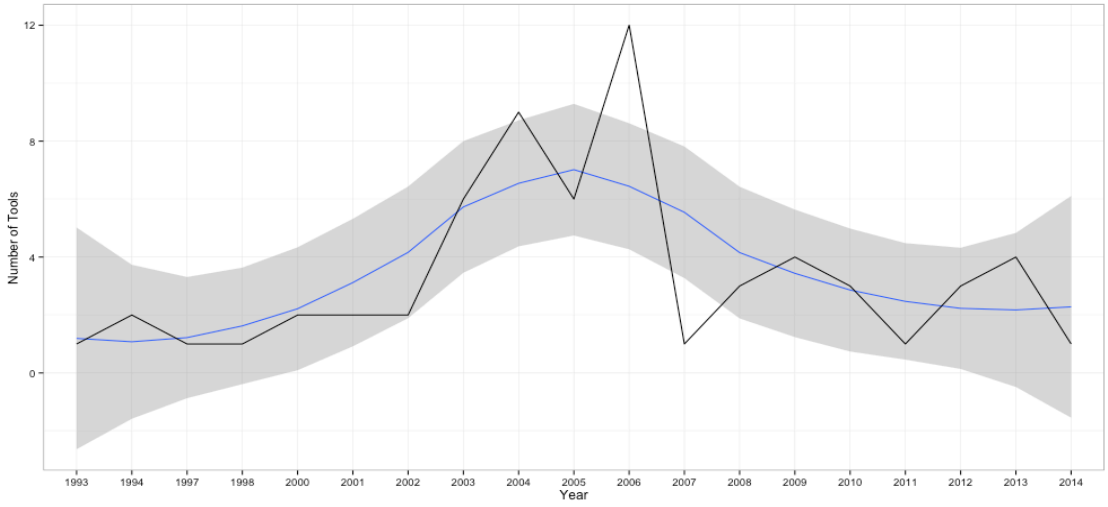
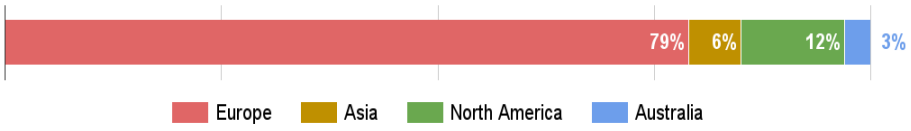
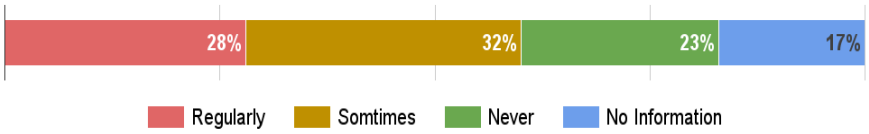


Figure 7: Number of new tools per year



(a) Number of tools based on the location



(b) Update time of tools

Figure 8: Model transformation tools

model transformation/code generation, but also verification and validation tasks. According to the figure, the majority of the tools, 39, do not include verification and validation functions. It has been mentioned that the models are *first-class citizens*, and model transformations are the *heart and soul* of MDE. Therefore, the quality and reliability of software development process depends on the quality of the models and model transformations. Mathematical basis of graph transformation tools can be used in formal verification tasks, such as the correctness of transformations. However, non-deterministic selection, fix-point iteration scheduling with concurrent application makes transformations not to be always confluent and terminate. Some tools can not guarantee the correctness or completeness properties, but support some static checks to validate and verify models. Tefkat supports the existence of the referred to meta-model elements, the basic well-formedness checking in the parser, and some editing supports to warn issues, such as variables only can be used once.

Considering reverse and round-trip engineering, non of the relational and visitor-based tools support these features, whereas some of imperative, 9, template-based, 10, and hybrid, 3, tools support reverse engineering.

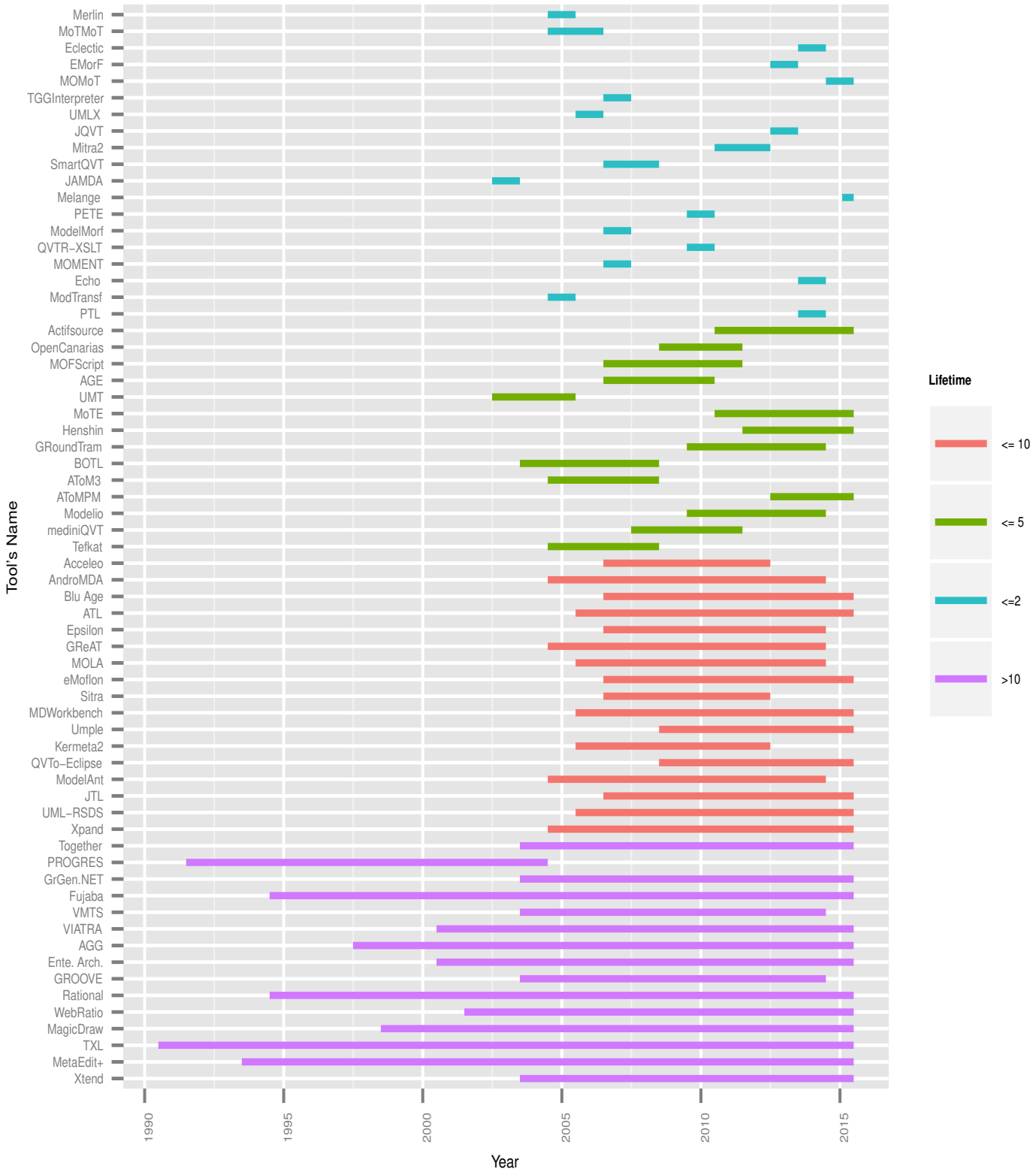


Figure 9: Survival lifetime of the tools

Figure 6 shows the data derived for *Implementation* and *Quality* facets such as tractability. As the traceability is one of the important features of MDD tools, the number of imperative tools which support built-in traceability outnumber the tools of the other approaches. Relations and operational mappings languages support automatic traceability, while traceability links must be handled manually in the QVTc. In graph-based tools there is no need to a distinguished separation between source and target models, so they are suited for in-place transformations. This feature makes it is hard to track traceability links between source and transformed elements to the target. Re-usability, incremental updates, concurrent transformation, and model comparison are useful when working with large and complex models. In that way, imperative, template-base and hybrid tools provide these features more than other tools.

8 Discussion of Maturity of Model-driven Development

In Figure 7, we map the graph gained based on the number of new tools in each year, with the graph of the diffusion of innovations proposed by Rogers [150]. The diffusion of innovations shows with successive groups of adopters of a new technology, its usage will eventually reach the saturation level. Figure 7 also demonstrates the same trend, so that there has been a clear increase in the number of new developed tools over the years of 2003 to 2007. However this trend has been declined and remained relatively stable since 2009. The questions that can arise are, what can it mean? Does it demonstrate the end of model-driven lifetime? This interpretation can be justified by this fact that MDD can make extra cost and efforts in terms of training new skills, and defining new roles. In addition to requiring to the knowledgeable employees, the task of designing and analyzing models may cause the overhead for software development process. We also looked at the number of tools developed in each continent. According to the Figure 8-a, 79% of tools have been developed in European countries in comparison to 12% tools in North America and 6% tools in Asia. It seems the model-driven engineering dose not receive enough support from different places over the world. Thus, MDD has not been accepted as a universal solution in the software development process. It is also possible to look at Figure 7 from another aspect that MDD area reaches to a stable phase. Considering Figure 8-b it may be a good idea to examine why some tools, such as ArcStyler, YATL, Codagen Architect, OptimalJ, and FUUT-je, along with 15 of the assessed tools (we do not have information about the update time of 11 of the tools which probably most of them will not have a new release) have not maintained no longer. For example, ModTransf has been developed before QVT standard for embedded systems. Its authors has stopped development with the availability of QVT-based tools. In addition, Figure 9 shows that 19 tools with the survival time <= 5 years, their last release were before 2013. Among the tools with survival time => 5, the last release of four tools are before 2013. TXL, and Metaedit+ have the longest survival time, and the shortest one is related to Melange. Based on these examinations, the reasons of success of long-time tools or the hidden issues can be found and analyzed.

9 Related Work

To best of our knowledge, there has not been a complete property-based comparison of the tools similar to this study. There are several surveys and frameworks to assess the abilities of some of the MDD tools. For example, there is a feature-based framework for the classification of model transformation approaches proposed by Czarnecki *et al.* [81]. They surveyed and classified model transformation techniques and tools into model-to-text (M2T) approaches, and model-to-model (M2M) approaches. Although their work is an excellent overview of the hierarchical classification of model transformation approaches, they have a heavier focus on transformation rules, which is just one aspect of model transformations. Mens *et al.* [82] have proposed a classification of model transformations tools that consider several factors, such as quality requirements for a transformation tool, verifying and guaranteeing correctness of the transformations in a multi-dimensional approach. Another interesting study has been conducted by Jakumeit *et al.* [84] which provides a detailed picture of 13 model and graph transformation tools participated at the *TTC* 2011. They compare the tools based on different factors, such as suitability (what is the tool suited for?), and data (which data is to be transformed).

Jilani *et al.* [136] use some analysis parameters, such as direction and understandability to survey some M2M transformation techniques. Another taxonomy proposed by Eramo *et al.* [137] focus on a set of relevant properties pertaining to bi-directional transformations. Similarly Hidaka *et al.* [138] propose a feature-based approach to compare different bi-directional model transformation approaches. Hildebrandt *et al.* [139]

propose a survey and a comparative study of TGGs tools. They emphasize on the necessity of correctness and the completeness for certain classes of the TGGs. There are other related works, such as [142, 140, 141] that focus on reviewing model transformation approaches/tools.

10 Conclusion

We have conducted a systematic literature review of classification and comparison of 65 tools based on the features which would contribute to effective usability of a tool. The features consist of six tables related to different aspects of the tool's features including, model-level, transformation language, capability, implementation and quality. Based on the gathered information, we analyzed different properties of the tools from different categories. We also evaluated how well the various MDD tools may perform to manage large and complex transformations.

The realm of model tools consists of different types, which means some features can only apply to certain kinds of tools, while are not applicable to others. However, we have shown that it is possible to consider important criteria of different aspects of a modeling tool, which can help the user(s) to select the best model transformation tool based on the their needs.

Acknowledgments

The authors would like to thank the authors who assisted in verifying the information on their tool's features. We also thank the anonymous reviewers for their valuable comments and suggestions in improving this report.

References

[1] Lano, K., and Kolahdouz-Rahimi, S. (2010, January). Specification and verification of model transformations using UML-RSDS. *In Integrated Formal Methods* (pp. 199-214). URL: `http://www.dcs.kcl.ac.uk/staff/kcl/uml2web`. Developed by: King's College London.

[2] Lawley, M., and Steel, J. (2006, January). Practical declarative model transformation with Tefkat. *In Satellite Events at the MoDELS 2005 Conference* (pp. 139-150). URL: `http://tefkat.sourceforge.net`. Developed by: CRC for Enterprise Distributed Systems (DSTC).

[3] Cicchetti, A., Di Ruscio, D., Eramo, R., and Pierantonio, A. (2011). JTL: a bidirectional and change propagating transformation language. *In Software Language Engineering* (pp. 183-202). URL: `http://jtl.di.univaq.it/index.php`. Developed by: University of L'Aquila.

[4] Almendros-Jiménez, J. M., Iribarne, L., López-Fernández, J., and Mora-Segura, A. (2015). PTL: A model transformation language based on logic programming. *Journal of Logical and Algebraic Methods in Programming*. URL: `http://indalog.ual.es/mdd/ptl2`. Developed by: University of Almeria and University of Madrid.

[5] Bonde, L., Dumoulin, C., and Dekeyser, J. L. (2005). Metamodels and MDA transformations for embedded systems. *In Advances in design and specification languages for SoCs* (pp. 89-105).

[6] Macedo, N. and Cunha, A. (2013). Implementing QVT-R bidirectional model transformations using alloy. *In the proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering*

[7] Boronat, A. (2007). MOMENT: a formal framework for MOdel manageMENT. *PhD Thesis in Computer Science*. URL: `http://moment.dsic.upv.es/content/view/34/75`. Developed by: Polytechnic University of Valencia.

[8] Li, D., Li, X., and Stolz, V. (January 2011). QVT-based model transformation using XSLT. *SIGSOFT Software Engineering Notes* 36:18. URL: `http://rcos.mydreamy.net/rcos-old/index.php?option=com_content&view=article&id=91:qvtr-xslt&catid=46:qvt-xslt&Itemid=104`. Developed by: University of Macau.

[9] Reddy, S., Venkatesh, R., and Ansari, Z. (2006). A relational approach to model transformation using QVT Relations. *TATA Research Development and Design Centre*. URL: `http://121.241.184.234/trddc_website/ModelMorf/ModelMorf.htm`.

[10] medini QVT. URL: http://projects.ikv.de/qvt/wiki. Developed by: *ikv++*

[11] Kelly, S., Lyytinen, K., and Rossi, M. (1996, January). Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In Advanced Information Systems Engineering (pp. 1-21). Springer Berlin Heidelberg. URL: http://www.metacase.com, Janne Luoma. Developed by: *MetaCase Consulting and University of Jyväskylä*, 2015.

[12] Schätz, B. (2009). Formalization and rule-based transformation of EMF Ecore-based models. *In Software Language Engineering* (pp. 227-244). URL: http://www4.informatik.tu-muenchen.de/~schaetz/PETE/PETEFrame.html. Developed by: *Technical university of Munchen.*

[13] Popov, R. (2015). ModelAnt. URL: http://mdatools.net/blog/modelant.

[14] Efftinge, S., (2015). Xtend. URL: https://eclipse.org/xtend/index.html. Developed by: *openArchitectureWare- Eclipse M2T*

[15] Gerking, C., and Heinzemann, C. (2014). Solving the Movie Database Case with QVTo. *TTC*. URL: URL:http://wiki.eclipse.org/QVTo. Developed by: *Eclipse Model-to-Model Transformation (MMT) project.*

[16] Drey, Z., Faucher, C., Fleurey, F., Mahé, V., and Vojtisek, D. (2009). Kermeta language reference manual. URL: http://www.kermeta.org. Developed by: *University of Rennes, and Triskell Team*

[17] Desfray, P. (2015). Modelio/Objecteering. *Modeliosoft*. URL: http://www.modeliosoft.com. Developed by: *Modeliosoft.*

[18] Forward, A., Lethbridge, T. C., and Brestovansky, D. (2009, May). Improving program comprehension by enhancing program constructs: An analysis of the Umple language. *In ICPC*. (pp. 311-312). URL: http://www.umple.org. Developed by: *University of Ottawa.*

[19] Capelle, T. (2015). MDWorkbench. URL: http://sodius.com/products-overview/mdworkbench. Developed by: *Sodius SAS.*

[20] Degueule, T., Combemale, B., Blouin, A., Barais, O., and Jézéquel, J. M. (2015, October). Melange: A Meta-language for Modular and Reusable Development of DSLs. In 8th International Conference on Software Language Engineering (SLE). URL: http://melange-lang.org. Developed by: *DiverSE research team-INRIA Triskell team.*

[21] Mazeika, D. (2015). MagicDraw. URL: http://www.nomagic.com. Developed by: *No Magic, Inc.*.

[22] Boocock, P. (2003). Jamda: the Java Model Driven Architecture. URL: http://jamda.sourceforge.net/#documentation

[23] IBM Rational Rose Family. URL: http://www-03.ibm.com/software/products/en/ratirosefami. Developed by: *International Business Machines (IBM) Corporation.*

[24] OBryan, D. (2015). Enterprise Architect. URL: http://www.sparxsystems.com. Developed by: *Sparx Systems*

[25] Blu Age. URL: http://www.bluage.com/en/en_home.html. Developed by: *Netfective Technology*

[26] Sánchez-Barbudo, A., Sánchez, E. V., Roldán, V., Estévez, A., and Roda, J. L. (2008). Providing an open virtual-machine-based QVT implementation. *In Proceedings of the V Workshop on Model-Driven Software Development*. URL: http://www.modelset.es/atc/atcdownload.html. Developed by: *Open Canarias.*

[27] Dupe, G., Belaunde, M., Perruchon, R., Besnard, H., Guillard, F., and Oliveres, V. SmartQVT. URL: https://sourceforge.net/projects/smartqvt. Developed by: *France Telecom R&D*

[28] Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Van Mierlo, S., and Ergin, H. (2013). AToMPM: A Web-based Modeling Environment. *In Demos/Posters/Student Research MoDELS* (pp. 21-25). URL: http://www-ens.iro.umontreal.ca/~syriani/atompm/atompm.htm. Developed by: *University of McGill, University of Montreal, and University of Antwerp.*

[29] Rensink, A. (2004). The GROOVE simulator: A tool for state space generation. *In Applications of Graph Transformations with Industrial Relevance* (pp. 479-485). URL: `http://groove.cs.utwente.nl/downloads/groove`. Developed by: *University of Twente.*

[30] Willink, E. D. (2003, June). UMLX: A graphical transformation language for MDA. *In Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications* (pp. 13-24). URL: `https://www.eclipse.org/gmt/umlx/-http://www.eclipse.org/projects/archives.php`

[31] De Lara, J., and Vangheluwe, H. (2002, April). AToM3: A tool for multi-formalism and meta-modelling. In FASE (Vol. 2, pp. 174-188). URL: `http://atom3.cs.mcgill.ca`. Developed by: *University of McGill*

[32] De Lara, J., Vangheluwe, H., and Alfonseca, M. (2004). Meta-modelling and graph grammars for multi-paradigm modelling in AToM3. *Software and Systems Modeling* (pp.194-209).

[33] Ermel, C., Rudolf, M., and Taentzer, G. (1999). The AGG approach: Language and environment. *Handbook of graph grammars and computing by graph transformation* (pp. 551-603). URL: `http://www.tfs.tu-berlin.de/agg`. Developed by: *Technical University of Berlin*

[34] Braun, P., and Marschall, F. (2003). Transforming object oriented models with BOTL. *Electronic Notes in Theoretical Computer Science* (pp. 103-117). URL: `http://botl.sourceforge.net`.

[35] Hidaka, S., Hu, Z., Inaba, K., Kato, H., and Nakano, K. (2011, November). GRoundTram: An integrated framework for developing well-behaved bidirectional model transformations. *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 480-483). URL: `http://www.biglab.org`. Developed by: *BiG team of the National Institute of Informatics*

[36] Lauder, M., Anjorin, A., Varró, G., and Schürr, A. (2012). Bidirectional model transformation with precedence triple graph grammars. *In Modelling Foundations and Applications* (pp. 287-302). URL: `http://www.moflon.org/emoflon`. Developed by: *Technical University of Darmstadt.*

[37] Arendt, T., Biermann, E., Jurack, S., Krause, C., and Taentzer, G. (2010). Henshin: advanced concepts and tools for in-place EMF model transformations. *In Model Driven Engineering Languages and Systems* (pp. 121-135). URL: `https://www.eclipse.org/henshin`. Developed by: *Philipps University Marburg, Technical University of Berlin and CWI Amsterdam*

[38] Giese, H., Hildebrandt, S., and Lambers, L. (2014). Bridging the gap between formal semantics and implementation of triple graph grammars. *Software and Systems Modeling* (pp. 273-299). URL: `www.mdelab.de/mote`. Developed by: *Hasso-Plattner Institute, and University of Potsdam*

[39] Kalnins, A., Barzdins, J., and Celms, E. (2005). Model transformation language MOLA. *In Model Driven Architecture* (pp. 62-76). URL: `http://mola.mii.lu.lv`. Developed by: *University of Latvia.*

[40] GReAT. URL: `http://www.isis.vanderbilt.edu/tools/great`. Developed by: *University of Vanderbilt.*

[41] Greenyer, J., and Kindler, E. (2007). Reconciling TGGs with QVT. *In Model Driven Engineering Languages and Systems* (pp. 16-30). URL: `http://www.cs.uni-paderborn.de/index.php?id=12842&L=1`. Developed by: *University of Paderborn*

[42] Klassen, L., and Wagner, R. (2012). EMorF-A tool for model transformations. *Electronic Communications of the EASST* 54. URL: `http://www.emorf.org/overview.html`. Developed by: *Solunar GmbH.*

[43] Schürr, A., Winter, A. J., and Zündorf, A. (1999). The PROGRES approach: Language and environment. URL: `http://www-i3.informatik.rwth-aachen.de/tikiwiki/tiki-index.php%3Fpage=Research:+Progres.html`. Developed by: *University of Technology Aachen.*

[44] Oldevik, J. UMT-UML URL: `http://umt-qvt.sourceforge.net/`.

[45] Varró, D., and Balogh, A. (2007). The model transformation language of the VIATRA2 framework. *Science of Computer Programming* (pp. 214-234). URL: `http://eclipse.org/viatra`. Developed by: *Budapest University of Technology and Economics (BUTE) and IncQuery Labs Ltd..*

[46] Cuadrado, J. S. (2012). Towards a family of model transformation languages.*In Theory and Practice of Model Transformations* (pp. 176-191). URL: `http://sanchezcuadrado.es/projects/eclectic/`.

[47] Kolovos, D. S., Paige, R. F., and Polack, F. A. (2008). The epsilon transformation language. *In Theory and practice of model transformations* (pp. 46-60). URL: `http://www.eclipse.org/epsilon/`. Developed by: *University of York*.

[48] Cuadrado, J. S., Molina, J. G., and Tortosa, M. M. (2006, January). Rubytl: A practical, extensible transformation language. *In Model Driven ArchitectureFoundations and Application*s (pp. 158-172). URL: `http://gts.inf.um.es/trac/age`. Developed by: *miso research group*

[49] Levendovszky, T., Lengyel, L., Mezei, G., and Charaf, H. (2005). A systematic approach to metamodeling environments and model transformation systems in VMTS. *Electronic Notes in Theoretical Computer Science* (pp. 65-75). URL: `https://www.aut.bme.hu/Pages/Research/VMTS/Introduction`. Developed by: *Budapest University of Technology and Economics*

[50] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Science of computer programming* (pp. 31-39). URL: `http://www.eclipse.org/atl/`. Developed by: *OBEO and INRIA, LINA research group*

[51] Van Gorp, G., (2008). Model-driven Development of Model Transformations. *PhD Thesis, University of Antwerp*. URL: `http://www.fots.ua.ac.be/motmot/index.php`. Developed by: *University of Antwerp*.

[52] Barais, O., Baudry, B., Blouin, A., Combemale, B., Jézéquel, J. M., and Vojtisek, D. (2013, April). A Demonstration for Building Modular and Efficient DSLs: The Kermeta v2 Experience. *In Conference of IngénieriE du Logiciel (CIEL)*.

[53] Degueule, T., Combemale, B., Blouin, A., and Barais, O. (2015, October). Reusing Legacy DSLs with Melange. *In 15th Workshop on Domain-Specific Modeling*.

[54] Dumoulin, C. (2004). ModTransf: a model to model transformation engine. URL: `http://www.lifl.fr/west/modtransf/`.

[55] Brun, C., and Pierantonio, A. (2008). Model differences in the eclipse modeling framework. *The European Journal for the Informatics Professional* (pp. 29-34). URL: `www.acceleo.org`. Developed by: *Obeo*.

[56] Cuadrado, J. S., and Molina, J. G. (2007). Building domain-specific languages for model-driven development. *Software, IEEE* (pp. 48-55).

[57] Anjorin, A., Lauder, M., Patzina, S., and Schürr, A. (2011). eMoflon: Leveraging EMF and Professional CASE Tools. *Informatik*.

[58] Ermel, C., Biermann, E., Schmidt, J., and Warning, A. (2011). Visual modeling of controlled emf model transformation using Henshin. *Electronic Communications of the EASST*.

[59] Bohlen, M., and Brandon, C. (2007). AndroMDA. URL: `http://www.andromda.org`. Developed by: *AndroMDA*.

[60] Carrara, R. (2015). URL: `http://www.actifsource.com`. Developed by: *actifsource GmbH*.

[61] Xpand. URL: `https://eclipse.org/modeling/m2t/?project=xpand`. Developed by: *openArchitectureWare and Eclipse M2T*.

[62] Lengyel, L., Levendovszky, T., Mezei, G., and Charaf, H. (2006). Model transformation with a visual control flow language. *International Journal of Computer Science (IJCS)* (pp. 45-53).

[63] Nickel, U., Niere, J., and Zündorf, A. (2000, June). The FUJABA environment. *In Proceedings of the 22nd International Conference on Software Engineering* (pp. 742-745). URL: `http://www.fujaba.de`. Developed by: *University of Paderborn*.

[64] Niere, J., and Zündorf, A. (2000). Testing and simulating production control systems using the Fujaba environment. *In Applications of Graph Transformations with Industrial Relevance* (pp. 449-456).

[65] Paige, R., and Radjenovic, A. (2003). Towards model transformation with TXL. *Metamodelling for MDA*. URL: `http://www.txl.ca`. Developed by: *University of Queen's*.

[66] Cordy, J. R. (2006). The TXL source transformation language. *Science of Computer Programming* (pp. 190-210).

[67] Jakumeit, E., Buchwald, S., and Kroll, M. (2010). Grgen.net: the expressive, convenient and fast graph rewrite system. *International Journal on Software Tools for Technology Transfer* (pp. 263-271).

[68] Jakumeit, E., Blomer, J., and Geiß, R. (2015). The GrGen.NET User Manual. URL: `http://www.grgen.net`. Developed by: *University of Karlsruhe*.

[69] Fleck, M., Troya, J., and 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). Elsevier. URL: `http://martin-fleck.github.io/momot/`.

[70] SiTra. URL: `http://baserg.github.io/sitra`. Developed by: *University of Kent*.

[71] Acerbis, R., Bongio, A., Brambilla, M., and Butti, S. (2007). Webratio 5: An eclipse-based case tool for engineering web applications. In Web Engineering (pp. 501-505). Springer Berlin Heidelberg. URL: `http://www.webratio.com/site/content/en/home`. Developed by: *WebRatio Srl and Polytechnic University of Milan*.

[72] Acerbis, R., Bongio, A., Butti, S., Ceri, S., Ciapessoni, F., Conserva, C., Fraternali, P., and Carughi, G. T. (2004). Webratio, an innovative technology for web application development. *In Web Engineering* (pp. 613-614).

[73] Pilgrim, V. J. (2011). Computerunterstützte Modelltransformationen. *PhD Thesis in Computer Science*. URL: `http://jpilgrim.github.io/mitra2/`.

[74] Song, H., and Kiegeland, J. (2013). JQVT. URL: `http://sourceforge.net/p/jqvt/wiki/Home/`.

[75] Cheuoua, J. (2015). Merlin. URL: `https://sourceforge.net/projects/merlingenerator/?source=navbar`.

[76] Reeves, G. (2015). Together. URL: `:http://www.borland.com/Products/Requirements-Management/Together`. Developed by: *Borland*.

[77] MOFScript. URL: `https://www.eclipse.org/gmt/mofscript`. Developed by: *SINTEF ICT* .

[78] Object Management Group. URL: `http://www.omg.org`.

[79] MOF Model to Text Transformation Language. URL: `http://www.omg.org/spec/MOFM2T`.

[80] Query/Views/Transformation Language (QVT). URL: `http://www.omg.org/spec/QVT`.

[81] Czarnecki, K., and Helsen, S. (2006). Feature-based survey of model transformation approaches.*IBM Systems Journal* (pp. 621-645).

[82] Mens, T., and Van Gorp, P. (2006). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science* (pp. 125-142).

[83] Salem, R. B., Grangel, R., and Bourey, J. P. (2008). A comparison of model transformation tools: Application for Transforming GRAI Extended Actigrams into UML Activity Diagrams. *Computers in Industry* (pp. 682-693).

[84] Jakumeit, E., Buchwald, S., Wagelaar, D., Dan, L., Hegedüs, ., Herrmannsdörfer, M., Horn, T., Kalnina, E., Krause, Ch., Lano, K., Lepper, M., Rensink, A., Rose, L., Wätzoldt, S., and Mazanek, S. (2014). A survey and comparison of transformation tools based on the transformation tool contest. *Science of computer programming* (pp. 41-99).

[85] , Taentzer G., Ehrig, K., Guerra, E., De Lara, J., Lengyel, L., Levendovszky, T., Prange, U., Varro, D., and Varró-Gyapay, S. (2005,). Model transformation by graph transformation: A comparative study. *In Proceedings Workshop Model Transformation in Practice, Montego Bay, Jamaica*.

[86] Hidaka, S., Tisi, M., Cabot, J., and Hu, Z. (2015). Feature-based classification of bidirectional transformation approaches. *Software and Systems Modeling* (pp. 1-22).

[87] Gomes, C., Barroca, B., and Amaral, V. (2014). Classification of model transformation tools: Pattern matching techniques. *In Model-Driven Engineering Languages and Systems* (pp. 619-635).

[88] Biehl, M. (2010). Literature study on model transformations. *Royal Institute of Technology*.

[89] QVTd. URL: `https://projects.eclipse.org/projects/modeling.mmt.qvtd`.

[90] Models Conference (2015). URL: `http://cruise.eecs.uottawa.ca/models2015`

[91] Roy, C. K., Cordy, J. R., and Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming* (pp. 470-495).

[92] Rothenberg, J., Widman, L. E., Loparo, K. A., and Nielsen, N. R. (1989). The nature of modeling. *Rand*.

[93] Unified Modeling Language (UML). URL: `http://www.uml.org`

[94] Aers, K. (2011). Graphiti and GMF compared: Revisiting the graph editor. *EclipseCon 2011, Santa Clara, California*.

[95] Efftinge, S., and Völter, M. (2006, October). oAW xText: A framework for textual DSLs. *In Workshop on Modeling Symposium at Eclipse Summit*.

[96] Kleppe, A. G., (2007). A language description is more than a metamodel.

[97] Bruneliere, H., Cabot, J., Jouault, F., and Madiot, F. (2010, September). 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). URL: `https://eclipse.org/MoDisco`.

[98] Schürr, A. (1995, January). Specification of graph translators with triple graph grammars. *In Graph-Theoretic Concepts in Computer Science* (pp. 151-163).

[99] XML Metadata Interchange (XMI). URL: `http://www.omg.org/spec/XMI`

[100] Tratt, L. (2005). Model transformations and tool integration. *Software and Systems Modeling* (pp. 112-122).

[101] Miller, J., and Mukerji, J. (2003). MDA Guide Version 1.0. 1, Object Management Group.

[102] Bottoni, P., Koch, M., Parisi-Presicce, F., and Taentzer, G. (2001). A visualization of OCL using collaborations. *In the Unified Modeling Language. Modeling Languages, Concepts, and Tools* (pp. 257-271).

[103] Canonical XMI. URL: `http://www.omg.org/spec/XMI/2.5.1`

[104] eXtensible Graph Markup and Modeling Language. URL: `http://cgi7.cs.rpi.edu/research/groups/pb/punin/publichtml/XGMML`

[105] Human Usable Textual Notation (HUTN). URL: `http://www.omg.org/spec/HUTN`

[106] Common Warehouse Meta-model (CWM). URL: `http://www.omg.org/spec/CWM`

[107] Diagram Definition Specification (DD). URL: `http://www.omg.org/spec/DD`

[108] Object Constraint Language. URL: `http://www.omg.org/spec/OCL`

[109] Java Meta-data Interface (JMI). URL: `http://www.oracle.com/technetwork/java/index.html`

[110] Graph eXchange Language (GXL). URL: `http://www.gupro.de/GXL`

[111] Business Process Model and Notation (BPMN). URL: `http://www.bpmn.org`

[112] Meta-Object Facility (MOF). URL: `http://www.omg.org/mof`

[113] Eclipse Modeling Framework (EMF). URL: `https://eclipse.org/modeling/emf`

[114] Kernel Meta Meta Model (KM3). URL: `https://wiki.eclipse.org/KM3`

[115] Graphical Editing Framework (GEF). URL: `https://eclipse.org/gef`

[116] NetBeans Meta-data Repository (MDR). URL: `https://netbeans.org`

[117] Peterson, J. L. (1981). Petri net theory and the modeling of systems.

[118] Brambilla, M., Cabot, J., and Wimmer, M. (2012). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering* (pp. 1-182).

[119] Lano, K., Kolahdouz-Rahimi, S., and Poernomo, I. (2012). Comparative evaluation of model transformation specification approaches. *International Journal of Software and Informatics* (pp. 233-269).

[120] Assmann, U. (2000). Graph rewrite systems for program optimization. *ACM Transactions on programming Languages and Systems (TOPLAS)* (pp. 583-637).

[121] Varró, D., Varró-Gyapay, S., Ehrig, H., Prange, U., and Taentzer, G. (2006). Termination analysis of model transformations by petri nets. *In Graph Transformations* (pp. 260-274).

[122] Ehrig, H., Ehrig, K., De Lara, J., Taentzer, G., Varró, D., and Varró-Gyapay, S. (2005). *Termination criteria for model transformation* (pp. 49-63).

[123] Mohagheghi, P., Dehlen, V., and Neple, T. (2008). Towards a tool-supported quality model for model-driven engineering. *In Proceeding 3rd Workshop on Quality in Modelling at MODELS.*

[124] Edapt. URL: `https://www.eclipse.org/edapt/documentation.php`.

[125] Rose, L. M., Kolovos, D. S., Paige, R. F., and Polack, F. A. (2010). Model migration with epsilon flock. *In Theory and Practice of Model Transformations* (pp. 184-198).

[126] Amrani, M., Lúcio, L., Selim, G., Combemale, B., Dingel, J., Vangheluwe, H., Traon, Y. L., and Cordy, J. R. (2012, April). A tridimensional approach for studying the formal verification of model transformations. *IEEE Fifth International Conference on Software Testing, Verification and Validation* (pp. 921-928).

[127] Rensink, A., Schmidt, Á., and Varró, D. (2004, January). Model checking graph transformations: A comparison of two approaches. *In ICGT* pp. 226-241.

[128] Varró, D., and Pataricza, A. (2003). Automated formal verification of model transformations. *CSDUML* (pp. 63-78).

[129] Kastenberg, H., and Rensink, A. (2006). Model checking dynamic states in GROOVE. *In Model Checking Software* (pp. 299-305).

[130] Fleurey, F., Steel, J., and Baudry, B. (2004, November). Validation in model-driven engineering: testing model transformations. *First International Workshop on Model, Design and Validation* (pp. 29-40).

[131] Auzi, A., Brzdi, J., Bievskis, J., erns, K., and Kalni, A. (1991). Automatic construction of test sets: Theoretical approach. *In Baltic Computer Science* (pp. 286-359).

[132] Ráth, I., Varró, G., and Varró, D. (2009). Change-driven model transformations. *IIn Model Driven Engineering Languages and Systems*(pp. 342-356).

[133] Stephan, M., and Cordy, J. R. (2013, February). A Survey of Model Comparison Approaches and Applications. *In Modelsward* (pp. 265-277).

[134] Louridas, P. (2006). Version control. Software. *IEEE* (pp.104-107).

[135] Taentzer, G. (2003) AGG: A graph transformation environment for modeling and validation of software. *In Lecture Notes in Computer Science* (pp. 446453).

[136] Jilani, A. A. A., Usman, M., and Halim, Z. (2010). Model Transformations in Model Driven Architecture. *IUniversal Journal of Computer Science and Engineering Technology* (pp. 50-54).

[137] Eramo, R., Marinelli, R., and Pierantonio, A. Towards a Taxonomy for Bidirectional Transformation.

[138] Hidaka, S., Tisi, M., Cabot, J., and Hu, Z. (2015). Feature-based classification of bidirectional transformation approaches. *Software and Systems Modeling* (pp. 1-22).

[139] Hildebrandt, S., Lambers, L., Giese, H., Rieke, J., Greenyer, J., Schäfer, W., Lauder, M., Anjorin, A., and Schürr., A. (2013). A survey of triple graph grammar tools. *In International Workshop on Bidirectional Transformations (Bx).*

[140] Varró, D., Asztalos, M., Bisztray, D., Boronat, A., Dang, D. H., Geiß, R., Greenyer, J., Gorp, P. V., Kniemeyer, O., Narayanan, A., Rencis, E., and Weinell, E. (2008). Transformation of UML models to CSP: A case study for graph transformation tools. *In Applications of Graph Transformations with Industrial Relevance* (pp. 540-565).

[141] Stevens, P. (2008). A landscape of bidirectional model transformations. *In Generative and transformational techniques in software engineering* (pp. 408-424).

[142] Leblebici, E., Anjorin, A., Schürr, A., Hildebrandt, S., Rieke, J., and Greenyer, J. (2014). A Comparison of Incremental Triple Graph Grammar Tools. *IElectronic Communications of the EASST.*

[143] Schätz, B. (2010). Verification of model transformations. *IElectronic Communications of the EASS.*

[144] Winkler, S., and Pilgrim, J. (2010). A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling (SoSyM)* (pp. 529-565).

[145] Ferry, N., Song, H., Rossini, A., Chauvel, F., and Solberg, A. (2014, December). Cloud MF: Applying MDE to tame the complexity of managing multi-cloud applications. *In Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing* (pp. 269-277).

[146] Ráth, I., Bergmann, G., Ökrös, A., and Varró, D. (2008). Live model transformations driven by incremental pattern matching. *In Theory and Practice of Model Transformations* (pp. 107-121).

[147] Cuadrado, J. S., Guerra, E., and de Lara, J. (2013). The program is the model: Enabling transformations run. time. *In Software Language Engineering* (pp. 104-123).

[148] Ahmad, M., Bruel, J. M., Laleau, R., and Gnaho, C. (2012). Using RELAX, SysML and KAOS for Ambient Systems Requirements Modeling. *Procedia Computer Science* (pp. 474-481).

[149] France, R. B., Bruel, J. M., and LarrondoPetrie, M. M. (1997). An integrated object-oriented and formal modeling environment. *Journal of Object-Oriented Programming.*

[150] Rogers, E. M. (2010). Diffusion of innovations. *Simon and Schuster.*

[151] Blouin, D., Plantec, A., Dissaux, P., Singhoff, F., and Diguet, J. P. (2014). Synchronization of models of rich languages with triple graph grammars: An experience report. *In Theory and Practice of Model Transformations* (pp. 106-121).

[152] Klatt, B. (2007). Xpand: A closer look at the model2text transformation language. *Language.*