Qualifying chains of transformation with coverage based evaluation criteria

Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, Alfonso Pierantonio

Department of Information Engineering Computer Science and Mathematics
University of L'Aquila, Italy
francesco.basciani@graduate.univaq.it

Abstract. In Model-Driven Engineering (MDE) the development of complex and large transformations can benefit from the reuse of smaller ones that can be composed according to user requirements. Composing transformations is a complex problem: typically smaller transformations are discovered and selected by developers from different and heterogeneous sources. Then the identified transformations are chained by means of manual and error-prone composition processes. Based on our approach, when we propose one or more transformation chains to the user, it is difficult for him to choose one path instead of another without considering the semantic properties of a transformation.
In this paper when multiple chains are proposed to the user, according to his requirements, we propose an approach to classify these suitable chains with respect to the coverage of the metamodels involved in the transformation. Based on coverage value, we are able to qualify the transformation chains with an evaluation criteria which gives as an indication of how much information a transformation chain covers over another.

1 Introduction

Model-driven engineering (MDE) is a software discipline that employs models for describing problems in an application domain by means of metamodels. Different abstraction levels are bridged together by automated transformations which permit source models to be mapped to target models. In MDE, model transformations play a key role and in order to enable their reusability, maintainability, and modularity, the development of complex transformations should be done by composing smaller ones [1]. The common way to compose transformations is to chain them [2,1,3,4,5], i.e., by passing models from one transformation to another. This process can be supported by an infrastructure based on a graph representation able to calculate the possible transformation compositions going from one model to another. Technically the entire process is supported by a repository of models, metamodels and model transformations previously presented in [6]. This automatic process can be called *transformation chaining* and has been treated in [7]. Moreover the possible chains outcome of the discovery in the model transformations stored in the repository, could be more than one and the user is responsible for choosing the better one for its purpose. Beyond the metamodel compatibility property, selecting and chaining model transformations can involve also other

properties like information loss, frequency of use, user rating or metamodel coverage. All those possible properties could be considered in a model transformation process in order to facilitate the user in the selection phase, when more than one suitable chains are proposed to the user, and one of those must be selected.

Our work extends the approach defined in [7], to support the automatic discovery and composition of model transformations with respect to user requirements. In particular developers provide the system with the source models and specify the target metamodel, by relying on a repository of model transformations, all the possible transformation chains are calculated and proposed to the user. The extension offered here comprehends a mechanism to provide a chain transformation evaluation index in order to guide the user in the chain transformation selection. To this end we define two properties (that in Section 3 we call *Coverage Input* and *Coverage Output*) on the transformations which allow us to understand how much information is preserved within a transformation: with this information we provide a selection criterion when we are dealing with multiple chains. In this paper, we highlight the process able to calculate the metamodel *coverage*, that will be proposed to the user when more than one possible transformation chains are are pulled out.

This paper starts with a background section where the chaining mechanism is explained and in Section 3 we propose a chaining classification based on coverage while providing its formal definition. In Section 4 we explore the problem of having multiple chains in response to the user requests showing a scenario in which we use the coverage criteria to qualify them. Related works in Section 5 and Section 6 concludes the paper.

2 Model transformation chaining: background

Composing model transformations is a difficult problem that can be approached in two different ways [5]: by chaining separate model transformations and passing models from one transformation to another (*external composition*), or by composing two model transformation definitions into a new model transformation (*internal composition*). Even though both methods for composing transformations are important and complement each other, in this paper we focus on external composition[1]. Figure 1

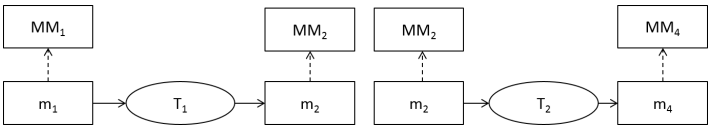

Fig. 1. Model transformation chain example

shows an explanatory model transformation chain. In particular, T_1 is a model transformation that generates models conforming to the target metamodel $\mathtt{MM}_2$ from models conforming to $\mathtt{MM}_1$. Additionally, T_2 is a model transformation that generates models conforming to $\mathtt{MM}_4$ from models conforming to the source metamodel $\mathtt{MM}_2$. Since the input metamodel of T_2 is also the output metamodel of T_1, then these two transformations

[1] For readability reasons, hereafter with the term *composition* we refer to *external composition*. Moreover, the terms *composition* and *chaining* are used interchangeably.

can be chained. Over the last years, different approaches have been studied to support the composition of model transformations (e.g., see [2,3,4,5]). The main activities that are typically performed when chaining model transformations are summarized in the following:

- *Specification of model transformation chains:* in this activity by considering the locally available model transformations, chains are specified by means of dedicated languages. For instance, in [8] the authors propose Wires*, a domain-specific language for the specification and orchestration of ATL transformations only. Another common way to chain model transformations is to use ANT scripts [2,3,4]. In [9] the authors propose the adoption of feature models to support the design of model transformation chains. In such a work, transformations are considered as features that are properly composed as specified in the considered feature models.
- *Execution of the specified model transformations chains:* in this phase the chains previously specified are executed on the source models given by the user. The execution environments of the adopted transformation languages are employed.

The first activity is the most complex one and over the last years a number of works have been presented to support it. The focus was mainly on the following aspects:

- *pre-* and *post-conditions* of transformations: when chaining transformations the conditions of applicability of a transformation (pre-conditions) and the conditions of validity of the resulting transformation (post-conditions) have to be satisfied. In [10] the authors propose an approach which adopting Higher-Order Transformations (HOT), is able to discover hidden chaining constraints between endogenous transformations by statically analysing the transformation rules.
- *commutativity/transformation order*: two model transformations are commutative (or parallel independent) if they can be chained in either order and produce the same results. In [1] the authors focus on this problem by providing an approach that permits to statically analyse two transformations and check if they are commutative or not.

In all the works mentioned above, the definition of transformation chains rely on the concept of *compatible metamodels* [2] as defined below.

Definition 1 (metamodels compatibility). *Let MM_1 and MM_2 be two metamodels, then MM_1 and MM_2 are* compatible *if $MM_1 \subseteq MM_2$ or $MM_2 \subseteq MM_1$.*

Definition 2 (transformation composability). *Let $T_1 : MM_1 \rightarrow MM_2$ be a model transformation from the metamodel MM_1 to the metamodel MM_2, and let $T_2 : MM_3 \rightarrow MM_4$ be a model transformation from the metamodel MM_3 to the metamodel MM_4. Then, T_1 and T_2 are* composable *as the sequential application $T_1;T_2$ if $MM_2 \subseteq MM_3$.*

[2] Apache Ant: http://ant.apache.org/
[3] Epsilon Workflow: http://www.eclipse.org/epsilon/doc/workflow/
[4] ATL-specific launch configurations and ANT tasks: http://wiki.eclipse.org/ATL/Howtos

The main focus of such works is to improve the solution found in like [1], in which authors check if two given transformations can be chained or not with respect to the metamodel compatibility property defined in [7]. Then, compatibility can be exploited to manually defining chains and singularly selecting the required transformations [9,8,3]. In [7] the authors, in response to user requests, define an automatic process in order to determine the transformation chains. Specifically, they define two activities *discovery of required model transformation (Discovery)* and *derivation of model transformation chains (Derivation)*, that by relying on a graph-based representation of a repository of artifacts [6] are able to determine all the paths between the source and target metamodels, i.e. all the possible chains that meet user requests.

3 Proposing chaining classification based on coverage

Figure 2 shows an extended version of the process seen in [7], by introducing the new activity ❷. In the previous process there were the activity ❶ (that provides a set of chains that transform a given model into a new one conform to the given target metamodel) and activity ❸ that executes the chosen chain.

The purpose of the new activity ❷ is to evaluate a specific transformation chain in order to facilitate the user chain selection. This evaluation is based on the amount of preserved information from the transformation. Therefore, with this new evaluation criteria we enrich the basic selection criteria for a chain with a new one: the coverage of the transformation with respect to the source and target metamodel.

In [11] Vignaga states that the coverage of a transformation, with respect to the source metamodel can be defined as the quotient between the total number of distinct source metaclasses whose instances are used in a transformation for producing the target models, and the total number of metaclasses in the source metamodels.

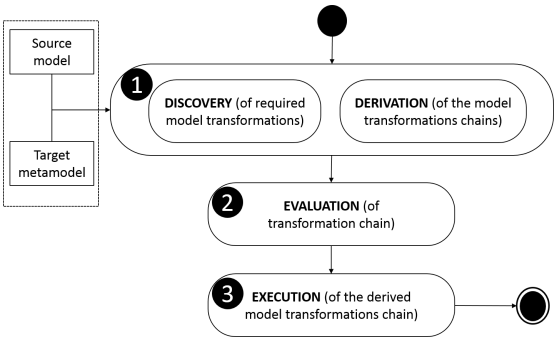

Fig. 2. Model transformations chaining process

Analogously we define the coverage of a model transformation with respect to its input metamodel (respectively output metamodel) as the ratio between the number of elements of the transformation that refer to the input metamodel (respectively output metamodel) with the total number of elements of the input metamodel itself (respectively metamodel output).

Metamodels / Transformation coverage checks which parts of a source (or target) metamodel are referenced by a given model transformation [12]. In this work, by analyzing the elements that compose the metamodels and transformations, we propose to use the value of coverage between the input and output metamodels and the transformation to classify the accuracy of the transformation we are choosing in the chaining

process. This is one of the possible criteria that can be used to evaluate transformations during the chaining process.

In the following we focus on how the coverage of an ATL model transformation is calculated with respect to its input and output metamodels. To obtain the *coverage values* (one for the input, *Coverage Input* , and one for the output, *Coverage Output*) the relationship between the number of elements of the input metamodel of the model transformation ($nElemInT(inputMM, T)$) (respectively output metamodel, $nElemInT(outputMM, T)$) on the number of elements of the input metamodel itself ($nElemInMM(inputMM)$) (respectively output metamodel, $nElemInMM(outputMM)$) is evaluated. More formally:

Definition 3 (Elements in a class). *Let $c \in C$ be a metaclass, let $nAttribute : C \to \mathbb{N}$, $nReference : C \to \mathbb{N}$, $nInheritsAttribute : C \to \mathbb{N}$, $nInheritsReference : C \to \mathbb{N}$ be functions that given a metaclass return the number of its attributes, the number of its references, the number of its inherits attributes and the number of its inherits references, respectively. Then $nElemInClass : C \to \mathbb{N}$ is defined as*

$$nElemInClass(c) = nAttribute(c) + nReference(c) + nInheritsAttribute(c) + nInheritsReference(c)$$

We are counting the number of elements in a class in terms of attributes and references (both inherited and non-inherited).

Definition 4 (Elements in a package). *Let P be a set of all packages and C a set of classes in metamodel. Let $p \in P$ be a package, let $classNotAbstract : C \to C', C' \subseteq C$, a function that given a package returns a set of its non-abstract class, let $subPackage : P \to P', P' \subseteq P$ be a function that given a package returns a set of its sub packages then $nElemInPackage : P \to \mathbb{N}$ is defined as*

$$nElemInPackage(p) = \sum_{s \in subPackage(p)} nElemInPackage(s) + \sum_{c \in classNotAbstract(p)} nElemInClass(c)$$

We are counting the number of elements in a package (and its sub-packages) and its non-abstract classes.

Definition 5 (Elements in a metamodel). *Let MM be a set of metamodels and let P be a set of packages. Let $mm \in MM$ be a metamodel, let $packageSet : MM \to P$ be a function that given a metamodel returns a set of its packages, then $nElemInMM : MM \to \mathbb{N}$ is defined as*

$$nElemInMM(mm) = \sum_{p \in packageSet(mm)} nElemInPackage(p)$$

We are counting the number of elements contained in a metamodel looking among its packages.

Definition 6 (Number of covered elements). *Let C be a set of metaclasses , let A be a set of attributes, let R a set of references, let T a set of transformations, let O a set of output pattern, let I be a set of input pattern, let B be a set of bindings, let $outPattern$:*

$T \rightarrow O$ be a function that given an transformation returns a set of all output pattern, let $inPattern : T \rightarrow I$ be a function that given an transformation returns a set of all input pattern, let $bindings : T \rightarrow B$ be a function that given a transformation returns a set of all bindings, let $referencedElement : B \rightarrow A \cup R$ be a function that given a binding returns a attribute or reference referenced by binding, let $t \in T$ be a transformation, let $mm_1 \in MM$ be a metamodels, let $isInMMClass : C, MM \rightarrow \{0, 1\}$ be a function that given a mataclass and a metamodel returns 1 if i are contained in MM, 0 otherwise, let $isInMMAttribute : A, C \rightarrow \{0, 1\}$ be a function that given an attribute and a metaclass returns 1 if i are contained in MM, 0 otherwise, let $isInMMRef : R, C \rightarrow \{0, 1\}$ be a function that given a reference and a metaclass returns 1 if i are contained in MM, 0 otherwise, let $distinctMetaclassInOp : T \rightarrow \{c \in C | \exists op \in outPattern(T) \wedge c = op.referredElements\}$ be a function that given a transformation returns a set of distinct metaclasses referenced by an out pattern, let $distinctMetaclassInIp : T \rightarrow \{c \in C | ip \in inPattern(T) \wedge c = op.referredElements\}$ be a function that given a transformation returns a set of distinct metaclasses referenced by an in pattern, let $distinctAttBindings : T \rightarrow \{a \in A \mid \exists b \in bindings(T) \ \wedge \ a = referencedElement(b)\}$ be a function that given a transformation returns a set of distinct attribute referenced by a binding, let $distinctRefBindings : T \rightarrow \{r \in R | \exists b \in bindings(T) \ \wedge \ r = referencedElement(b)\}$ be a function that given a transformation returns a set of distinct reference referenced by an binding then $nElemInT : T, MM \rightarrow \mathbb{N}$ is defined as

$$nElemInT : (mm_1, t) = \sum_{i \in distinctMetaclassInOp(T)} isInMMClass(i, MM) +$$

$$\sum_{i \in distinctMetaclassInIp(T)} isInMMClass(i, MM) +$$

$$\sum_{i \in distinctAttrInBindings(T)} isInMMAttribute(i, MM) +$$

$$\sum_{i \in distinctRefInBindings(T)} isInMMReference(i, MM)$$

We are counting how many elements of the metamodel, provided as input to the function (which can be either the source metamodel that the target metamodel), are covered by the transformation.

Definition 7 (Transformation coverage). *Let $mm_1 \in MM$ be a metamodels, let $t \in T$ be a transformation, then $tCoverage : (MM, T) \rightarrow [0, 1]$ defined as*

$$tCoverage(mm_1, t) = \frac{nElemInT(mm_1, t)}{nElemInMM(mm_1)}$$

With this formula we calculate the coverage of a transformation. Depending on whether we provide as input of the function the source or the target metamodel, we will have as a result, respectively, the input or output coverage.

How the coverage of a chain is calculated Starting from the graph in Fig. 3, representing our repository, and the inputs provided by the user (see Fig. 2) through a *breadth-first search (BFS)*, the system retrieves all the paths. Each retrieved path starting from the node that represents the metamodel provided as input, arrives at target metamodel still supplied by the user. It is important to remark that at this stage the coverage values on the edges are not yet considered.

Once the list of paths between source and target is obtained, the *chain coverage* is calculated and this is done through the formula derived from the following definition:

Definition 8 (Chain Coverage). *Let TC be a set of transformation chains, let MM be a set of metamodels, let T be a set of transformations, let $t \in T$ be a transformation, let $ct \in TC$ be a chain transformation, let $sourceMM : T \rightarrow MM$ be a function that given a transformation return the source metamodel, let $targetMM : T \rightarrow MM$ be a function that given a transformation return the target metamodel then $chainCoverage : TC \rightarrow [0, 1]$ is defined as*

$$chainCoverage(ct) = \prod_{t \in transformationChain(ct)} tCoverage(t, sourceMM(t)) * tCoverage(t, targetMM(t))$$

With this formula we take into account on the one hand the values of coverage in a chain and on the other hand we take into account the length of the same (having values between 0 and 1). Once we determined these values (for each insertion and/or deletion of a model transformation) they are retained as weights on edges in the graph structure (you can see an example in Fig. 3).

4 Dealing with multiple chains

The sub-activities *Discovery* and *Derivation* of activity ❶ in Fig. 2 might give place to different possible chains among which the user must choose.

and before going ahead with executing activity, users have to select one of them.

As said before, possible path of chaining can be distinguished based on different parameters and with this work we focus on the coverage of the transformation respect to the source and target metamodel. Based on the framework proposed in [6] and taking advantage of all its services in support of the chaining mechanism, what we do is to enhance the representation graph for the maintenance of the artifacts, adding values at the beginning (*Coverage Input*) and the end (*Coverage Output*) of an arc which represents a transformation.

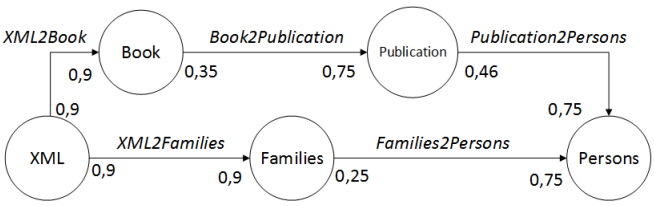

Fig. 3. Graph structure of metamodels and model transformations with *Coverage Input* and *Coverage Output* on edges

In other words an edge represents a transformation from source metamodel to the target in the repository, and the *coverage in input/output* represents the ratio between elements in the source/target metamodels and the elements consumed/produced by the transformation respectively.

Let us suppose that the user gives as input an *XML* model and requests to generate a target *Persons* model. According to the available transformations in the repository, the result of activity ❶ is the list of the chains, which in the example are these two:

(1) $XML2Families \rightarrow Families2Persons$
(2) $XML2Book \rightarrow Book2Publication \rightarrow Publication2Persons$

The model transformation *Families2Persons* is shown in Fig. 4.a, it is the second transformation composing the chain (1) that has been retrieved. This transformation has *Families* metamodel as source, represented in Fig. 4.b and *Persons* metamodel as target, depicted in Fig. 4.c.

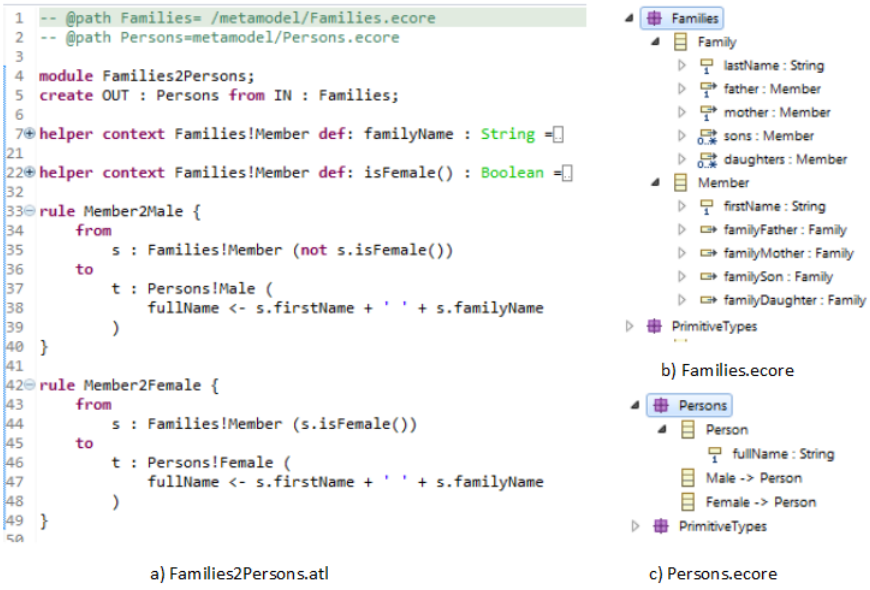

Fig. 4. Families2Persons example

This list of chains is processed by the activity ❷ (*Evaluation*). We calculate how much the model transformation actually ”covers” the elements of the source and target metamodel and we obtain Table. 1, summarizing the values that the system extrapolates by analyzing the source and target metamodels, and the model transformation.

The process to assign the coverage values to the existing transformations in the repository can be summarized as:

Metamodels static analysis → Transformation static analysis → Coverage Calculation

The system firstly extract from the static analysis of metamodels and transformation, the *Matched Rules* in the transformation and its related source and target patterns:

nElemInMM(Families)					
Classes	Abstracts	StructuralFeatures	InheritedStructuralFeatures	Attributes	References
Family	false	5*	0	1	4
Member	false	5**	0	1	4
Total			12		

nElemInMM(Persons)					
Classes	Abstracts	StructuralFeatures	InheritedStructuralFeatures	Attributes	References
Person	true	1***	0	1	0
Male	false	0	1	0	0
Female	false	0	1	0	0
Total			4		

nElemInT(Families, Families2Persons) / nElemInT(Persons, Families2Persons)				
	Classes (Not Abstract)	Attributes	References	Total Elements
Families	2	2	8	12
Persons	2	1	0	4

* *lastName, father, mather, sons, daughter*
** *firstName, familyFather, familyMother, familySon, familyDaughter*
*** *fullName*

Table 1. Report extrapolated by the system concerning the source and target metamodels and the model transformation.

– Source Classes: *Member*;
– Source Attributes covered by Matched Rules: *firstName*, *familyName*;
– Target Classes: *Male*, *Female*;
– Target Attributes: *fullName*;

than the system, according to the Definition 7 seen previously, outputs two *coverages values*, one for the source and one for the target:

$$TCoverage_{input} = \frac{3}{12} = 0,25 \qquad TCoverage_{output} = \frac{3}{4} = 0,75$$

Referring to the structural graph, representing our repository in Fig. 3, assuming that user requests as input metamodel XML and requires the target metamodel as $Persons$, the system derives all the possible chains with the following *coverage values*:

Transformation chain	Coverage Value
$XML2Families \rightarrow Families2Persons$	$0,15$
$XML2Book \rightarrow Book2Publication \rightarrow Publication2Persons$	0.07

The result that comes out from this activity of ”evaluation” suggest that could be convenient to invoke a chain like *XML2Families* $\rightarrow$ *Families2Persons*, that has a coverage value higher than the other one. After the user's selection, activity ❸ can start, i.e. the execution of the chain.

5 Related work

Increasingly, model transformation chaining has been a current topic of research and it has been treated from different perspectives.

In [2] the authors present a convenient approach to design highly flexible chains from existing independent model transformations. The main difference with the presented approach is the *design* part that in our case is automatically calculated by the engine in discovery mode. They propose to artificially change the input and output of transformations in order to recover the compatibility of the involved metamodels. The work in [2] is an extension of what is presented in [1] where the authors address the problem of identifying conflicts between transformations, and checking if two transformations are commutative or not.

A language for defining composition of transformations is given in [3]. To support the concrete realization of transformation chains they propose a language to allow the concatenation of transformation components. A recent work [9] uses feature models to classify model transformations. Based on these feature models, automated techniques help the designer to generate executable chain of transformations. Another interesting work has been exposed in [13] where transformation chaining is called *orchestration*. This paper introduces a graphical executable language for the orchestration of ATL transformations, which provides appropriate mechanisms to enable the modular and compositional specification and execution of complex model transformations chains. The work presented in [4] describes an approach to designing large model transformations for large languages, based on the principle of separation of concerns. Chains are built by linking output parameters to input ones through connectors. Differently from such works we do not require the specification of transformation chains that in our approach are automatically derived with respect to the request of the user and to the transformations, which are stored in a dedicated repository.

In [14] the authors propose a mechanism of module superimposition to compose small and reusable transformations. The idea is to overlay several transformation definitions on top of each other and then execute them as one transformation. Differently from our work, the approach in [14] is specific for ATL and it is an internal composition approach. The work proposed in this paper is an external composition technique and it is independent from the used model transformation language. Vignaga [11] describes a number of metrics for ATL model transformations, described according to the ATL metaclass to which they apply. In this paper and also in [12] the coverage is treated and the formula exposed has been reused also in our work.

6 Discussion and conclusions

In this paper we present an improvement of the approach seen in [7] that support model transformations chaining. Starting from a user request consisting of a source model, and the specification of a target metamodel, the system is able to calculate the possible chains satisfying the user request according to the transformation available in a proposed transformation repository. The main strengths of our approach are related to the possibility of qualifying the chains with the coverage that helps the user to choose

a chain among all the others that the system is able to retrieve. This is done by analyzing all the elements that compose both metamodels (input and target) of a model transformation and the model transformation itself. With this technique, in some way, we are going to evaluate how much information is preserved or lost when you make a single transformation and consequently how much it preserves or how much is lost in the whole chain. We are aware that in order to better characterize the concept of *information loss* we should consider in a different way both the models in input and output of a transformation. This, however, will be investigated in the next works.

References

1. Etien, A., Aranega, V., Blanc, X., Paige, R.F.: Chaining Model Transformations. In: Proceedings of the First Workshop on the Analysis of Model Transformations. AMT '12, New York, NY, USA, ACM (2012) 9–14
2. Etien, A., Muller, A., Legrand, T., Blanc, X.: Combining Independent Model Transformations. In: Proceedings of the 2010 ACM Symposium on Applied Computing. SAC '10, New York, NY, USA, ACM (2010) 2237–2243
3. Vanhooff, B., Van Baelen, S., Hovsepyan, A., Joosen, W., Berbers, Y.: Towards a Transformation Chain Modeling Language. In Vassiliadis, S., Wong, S., Hmlinen, T., eds.: Embedded Computer Systems: Architectures, Modeling, and Simulation. Volume 4017 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2006) 39–48
4. Etien, A., Muller, A., Legrand, T., Paige, R.F.: Localized model transformations for building large-scale transformations. Software Systems Modeling (2013) 1–25
5. Wagelaar, D.: Composition Techniques for Rule-Based Model Transformation Languages. In Vallecillo, A., Gray, J., Pierantonio, A., eds.: Theory and Practice of Model Transformations. Volume 5063 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 152–167
6. Basciani, F., Di Rocco, J., Di Ruscio, D., Di Salle, A., Iovino, L., Pierantonio, A.: MDEForge: an extensible Web-based modeling platform. CloudMDE 2014 (2014) 66
7. Basciani, F., Di Ruscio, D., Iovino, L., Pierantonio, A.: Automated chaining of model transformations with incompatible metamodels. In: Model-Driven Engineering Languages and Systems. Springer (2014) 602–618
8. Rivera, J.E., Ruiz-Gonzalez, D., Lopez-Romero, F., Bautista, J., Vallecillo, A.: Orchestrating ATL Model Transformations. In: Proc. of MtATL 2009, Nantes, France (2009) 34–46
9. Aranega, V., Etien, A., Mosser, S.: Using Feature Model to Build Model Transformation Chains. In: Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems. MODELS'12, Berlin, Heidelberg, Springer-Verlag (2012) 562–578
10. Chenouard, R., Jouault, F.: Automatically Discovering Hidden Transformation Chaining Constraints. In Schrr, A., Selic, B., eds.: Model Driven Engineering Languages and Systems. Volume 5795 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2009) 92–106
11. Vignaga, A.: Metrics for Measuring ATL Model Transformations. Technical report (2009)
12. Wang, J., Kim, S.K., Carrington, D.: Verifying metamodel coverage of model transformations. In: Software Engineering Conference, 2006. Australian. (2006) 10 pp.–
13. Rivera, J.E., Ruiz-Gonzalez, D., Lopez-Romero, F., Bautista, J., Vallecillo, A.: Orchestrating ATL Model Transformations. In: Proc. of MtATL 2009, Nantes, France (2009) 34–46
14. Wagelaar, D., Van Der Straeten, R., Deridder, D.: Module superimposition: a composition technique for rule-based model transformation languages. Software & Systems Modeling **9** (2010) 285–309