Towards Benchmarking Evolution Support in Model-to-Text Transformation Systems

Bernhard Hoisl Stefan Sobernig

Institute for Information Systems and New Media
Vienna University of Economics and Business (WU Vienna)
{bernhard.hoisl, stefan.sobernig}@wu.ac.at

Abstract

In model-driven development, an evolving metamodel as part of a changing software system requires the adaptation of interrelated artifacts, such as, model-to-text (M2T) transformation specifications. In this paper, we propose a definition for a standard problem to evaluate the evolution support in M2T transformation systems. The objective of the standard problem is to allow for benchmarking of multiple evolution-support techniques for M2T transformations. For this, we selected an existing, real-world software application acting as the basis for the standard-problem definition, describe a metamodel-evolution scenario (migration), and define a measurement plan to benchmark different implementations (thus, making them comparable). The applicability of the standard problem definition is exemplified by benchmarking an approach of higher-order rewriting M2T generator templates.

1 Introduction

In model-driven development (MDD), the domain-specific metamodel is typically created in several iterations and is therefore not a static artifact. For example, adaptations of domain requirements can trigger the evolution of a metamodel (see, e.g., [WKS+10, RIP12]). Requirements can change, for instance, due to additional functionality, a modified legal situation in the corresponding application domain, or the refactoring of software systems (see, e.g., [HSS14]). In such a metamodel-evolution scenario (see, e.g., [CP10, HSS12]), the interrelated artifacts shaping the *metamodeling ecosystem* (e.g., instance models, model transformations) must be adapted to apply to the evolved metamodel (a.k.a. "coupled evolution" [RIP12]).

In this paper, we take the challenge of coupled evolution for model-to-text (M2T) transformations caused by an adapted metamodel. There is evidence on M2T transformations targeting source code being the most widespread platform-integration technique in MDD [Gen10]. This is confirmed by a recent study on M2T transformations for UML-based domain-specific modeling languages [SHSed]. In particular, we focus on M2T generator templates [SV06]—as one commonly employed implementation technique for creating platform-specific artifacts. A metamodel evolution entails the adaptation of generator templates defined over the original metamodel to conform to the evolved metamodel (e.g., to reflect changed metamodel elements).

Metamodel evolution can fall along a spectrum when viewed from the perspective of resulting heterogeneity between the original and the evolved metamodels [WKK+10]. Forms of syntactic heterogeneity include naming differences and structural variations (e.g., different source-target cardinalities). Semantic heterogeneity results from differences in the interpretation of two metamodels (e.g., in the conditions rendering an instance model valid). Certain forms of (esp. syntactic) heterogeneity can be detected and/or resolved in an automated and tool-assisted manner (automated refactoring), others require manual inspection and intervention (manual refactoring).

Researchers have proposed different techniques to handle the coupled evolution problem of M2T generator templates in terms of automation support for refactorings of M2T generator templates in response to metamodel evolution. These techniques are higher-order transformations (HOTs [TJF+09, Hoi14]), generic templates, and adapter models (see, e.g., [HSS13]). For each approach, different (reference) implementations have been made available targeting different M2T transformation systems. Automation support for evolving M2T generator templates has a number of benefits: the reuse of M2T transformations is facilitated, concepts used in M2T transformations are matched automatically with their evolved metamodeling concepts, and the evolutionary process is explicitly documented (see, e.g., [Hoi14]).

Currently, however, we lack understanding of the conceptual strengths and weaknesses of the different techniques and the corresponding implementations regarding the spectrum of metamodel evolution. For example, the different techniques (HOTs, generic templates, adapter models) rely on certain assumptions on the permitted metamodel heterogeneities, which are often not stated explicitly. From a researcher's perspective, this hampers an analytical comparison making it difficult to answer questions, such as: Given an unanticipated metamodel evolution, known to involve syntactic heterogeneities only, which of the three techniques is sufficient to port existing M2T generators without incurring excessive extra (manual) effort? For practitioners, the choice of a technique (implementation) so becomes determined by convenience or by uninformed selection.

To render these techniques and their implementations comparable, we propose a standard problem for assessing strenghts and weaknesses of competing refactoring techniques for M2T generator templates. In addition, the proposed standard problem provides assets which serve as a benchmark for the non-functional properties of evaluated implementations (e.g. time and space efficiency). Related work has contributed standard problems and benchmarks, for instance, for language workbenches[1] and for model-to-model (M2M) transformation systems (see, e.g., [LW13, WKK+10, vdBHVP11]). To the best of our knowledge, there exists no such standard-problem definition for the coupled evolution of M2T generator templates.

Therefore, in this paper, we report on a first step towards defining and validating a standard problem for this particular kind of software evolution in MDD (such as [RRIP14, HSS13]). It is evident that such a standard problem cannot be formulated without the help and the feedback of the broader research community, including the AMT committee and the AMT participants. To this end, this paper should act as a stimulus for feedback and critical discussion. We believe that a final definition of this standard problem plus benchmark must have the following characteristics:

- It must be capable of hosting different classes of metamodel heterogeneity to reflect different metamodel-evolution settings.
- It should draw on common application-engineering knowledge to minimize the entry barriers for non-experts in the selected domain while representing a non-trivial evolution scenario.
- It should provide clear guidelines on how to apply the standard problem as well as on how to benchmark and on how to compare different implementations.

The remainder of the paper is structured as follows. We sketch the objectives of our proposed standard problem in Section 2. In Section 3, we present details on the definition of the standard problem. In particular, we explain the selected real-world application, the metamodel-evolution scenario as well as base and derived metrics. We demonstrate the applicability of our standard problem definition by benchmarking an example implementation in Section 4. Afterwards, we discuss our approach in Section 5 and related work in Section 6. At last, in Section 7, we conclude the paper and point to future work.

2 Objective: Benchmarking

The objective of the standard-problem definition is to enable a systematic and primarily quantitative comparison of one's technique (implementation) with alternative techniques (implementations), that is, *benchmarking*. The predefined details of the standard-problem definition (e.g., model artifacts, base metrics, benchmark score) turn the evaluation results obtained for one's technique into a benchmark for others. As relates to the content, the benchmarking aims at four quality dimensions of refactoring M2T transformations by collecting quantitative data on them (see, e.g., [FP97, LW13]): The a) *completeness* of the evolved concepts of the generator templates (i.e. how many concepts can automatically be refactored); the b) *correctness* of the produced platform-specific artifacts (i.e. source code) from the evolved generator templates; the c) *complexity* change of the evolved generator templates (compared to the original ones); and the d) *runtime performance* of evolving the generator templates.

[1] `http://www.languageworkbenches.net/` (last accessed on 2015-09-16).

To create a benchmark data set, quantitative data is to be collected according to a measurement plan defined using the goal-question-metric (GQM) method (see, e.g., [vSB99]). In GQM, goals are formulated first (conceptual level). Then, a set of questions is defined to characterize the way the assessment of a specific goal is going to be performed (operational level). At last, metrics[2] are specified and associated with every question in order to answer it in a quantitative way (quantitative level). Table 1 shows our GQM model for quantifying data according to our benchmark's criteria from the viewpoint of a transformation developer. This way, four questions are defined with corresponding metrics which all contribute to characterize a M2T transformation system under evolution. The metrics are described in detail in Section 3.3.

Table 1: GQM model for evaluating systems according to our proposed benchmark criteria.

Goal	Purpose	Quantifying and making comparable
	Issue	completeness, correctness, complexity, and performance criteria of
	Object	a software system to support the evolution of M2T generator templates
	Viewpoint	from the transformation developer's viewpoint.
Question		To what extent does a system reduce the need to manually perform refactorings on M2T generator templates (related to the completeness criterion)?
Metric		1. Number of automatically refactored M2T transformation concepts by using the system.
Question		To what extent does a system produce application-specific artifacts as intended (related to the correctness criterion)?
Metric		2. Number of errors in the generated software artifacts (i.e. source code).
Question		To what extent does a system reduce the complexity of evolved M2T generator templates (related to the complexity criterion)?
Metrics		3. Number of called helper operations. 4. Number of calls to helper operations. 5. Transformation size in source lines of code (SLOC).
Question		How long does a system need for the process of refactoring M2T transformation templates (related to the performance criterion)?
Metric		6. Execution time of the M2T-generator-template evolution.

3 A Proposal for a Standard Problem

3.1 Application Selection

To set a standard problem and the basis for benchmarking, the selected application must fulfill a number of requirements. First, it must include non-trivial M2T transformation definitions (generator templates) in terms of complexity regarding metamodel-element changes and the SLOC size of M2T transformation templates. Second, the measurement plan requires that the source-code base is fully available and can be processed automatically (i.e. metamodels, input models, M2T transformation templates). Third, we require that all artifacts are publicly available to make the benchmark generally applicable and its results reproducible.[3]

To find a suitable application, we reached out the MDD community, for example, via postings in relevant Eclipse sub-forums for hints and contacted research and industry peers. One of our colleagues at the Department of Computer Science at the University of York (Dimitris Kolovos) pointed us to the Pongo project [KW15]. Pongo describes itself as "a template-based Java POJO generator for MongoDB. Instead of using low-level `DBObjects` to interact with your MongoDB database, with Pongo you can define your data/domain model using EMFatic and then generate strongly-typed Java classes you can then use to work with your database at a more convenient level of abstraction" [KW15]. As the Pongo project fulfilled our requirements (non-trivial M2T transformation definitions, open-source, publicly available), we adopted it as basis for this first standard-problem definition.

The problem assets are derived from material (domain model, test application) obtained from the Pongo tutorial (a blogging system) published on the project's website [Kol15]. The domain model of the blogging system is specified in the EMFatic textual syntax. Figure 1 shows the equivalent Ecore model which defines four

[2] We use the terms "metric" and "measure" interchangeably for the scope of this paper.

[3] In turn, all software artifacts used in and developed for the standard problem as well as the benchmarking example in this paper can be obtained from `http://nm.wu.ac.at/modsec`.

`EClasses` (`Blog`, `Post`, `Comment`, and `Author`) as well as corresponding attributes and references to represent the blogging domain.[4]

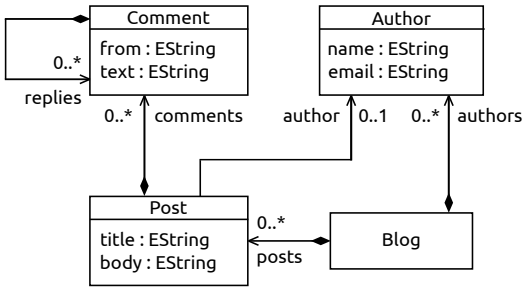

Figure 1: Ecore model of the blogging system.

The blogging system uses M2T transformations implemented in Epsilon (EGL templates with EOL helper operations and EGX as coordination language for EGL templates [KRGDP15]) to generate Java source code from Ecore models. By executing the transformations, six Java files are generated. These Java classes implement the domain model (see Figure 1) and define helper methods (e.g., getter and setter) to conveniently work with the MongoDB database (e.g., for reading and writing data). The Pongo tutorial provides a Java application used for testing the generated Java classes [Kol15]. If the M2T transformation from the Ecore-based domain model into Java classes is successful, the test application executes free of errors.

3.2 Metamodel-Evolution Scenario: Language and Platform Migration

Migrating a given, model-driven application from one metamodeling language and metamodeling platform to another is a frequently observed sample migration scenarion requiring metamodel evolution. Others are application improvement or replacement/rewrite scenarios (see [UN10] in more general). The migration scenario is typically driven by technology obsolescence (e.g., an inter-organizational technology standard being superseded by another one: UML 1.* to UML 2.* [RHM+14]) or by the desire to lift and shift a model-driven application to a (new) organizational standard (e.g. legacy metamodels [SWCD12]). This scenario does not involve any substantial redesign in terms of domain abstractions beyond what is required to find equivalent (metamodeling) concepts in the target language and platform. Unlike other scenarios, this scenario is, therefore, marked by a high potential of automation and of tight coupling between metamodel transformation and refactoring of related artifacts (M2T generator templates). We selected this particular scenario to form a standard problem because of its capability to capture automation potential and because it is a frequently adopted scenario in similar settings (e.g. MDD tool competitions [RHM+14]).

For the definition of the standard problem, we propose an instantiation of this migration scenario for Pongo and its M2T transformation artifacts.[5] The migration is one from porting Pongo to support UML2 class models besides Ecore models, therefore a migration from the Ecore metamodel to the UML2 metamodel. The underlying mappings of this migration scenario are specified as M2M transformation operations. To maximize adoptability, we have chosen the notation of technology-independent *mapping diagrams* from [GdLK+13]. Mapping diagrams provide "a high-level design view" [GdLK+13] on transformations, thereby abstracting from concrete transformation languages. Figure 2 shows the *Ecore2UML* transformation operations for our benchmark.

Note that the mapping is not exhaustive—it does not cover all details of the source and target metamodels (Ecore and UML2, respectively). Rather, the mapping reflects the necessary subset over which the M2T transformation definitions are defined (i.e. their model domains). For Pongo, we found that six transformation operations are required to represent the Ecore2UML metamodel evolution sufficiently to capture all parts of the M2T generator templates relevant for the refactoring (see Figure 2). These transformation operations form the benchmarking basis. They reflect different kinds of syntactical heterogeneity between the Ecore and UML metamodels, in particular, differences according to the source-target-concept cardinality (1:1 and n:1) as well as naming, multiplicity, containment, and context differences of the same metamodeling concept (`EReference`) [WKK+10].

[4] Please note that the Ecore model shown in Figure 1 corrects two errors in the original EMFatic textual definition [Kol15]: 1) A missing composite aggregation (`comments`) pointing from the `EClass Post` to the `EClass Comment` has been added and 2) the name of the attribute `body` owned by the `EClass Comment` has been replaced with `text` (see also [Hoi14]).

[5] The scenario instantiation was also inspired by the discussion at `https://www.eclipse.org/forums/index.php/t/488742/` (last accessed on 2015-09-16).

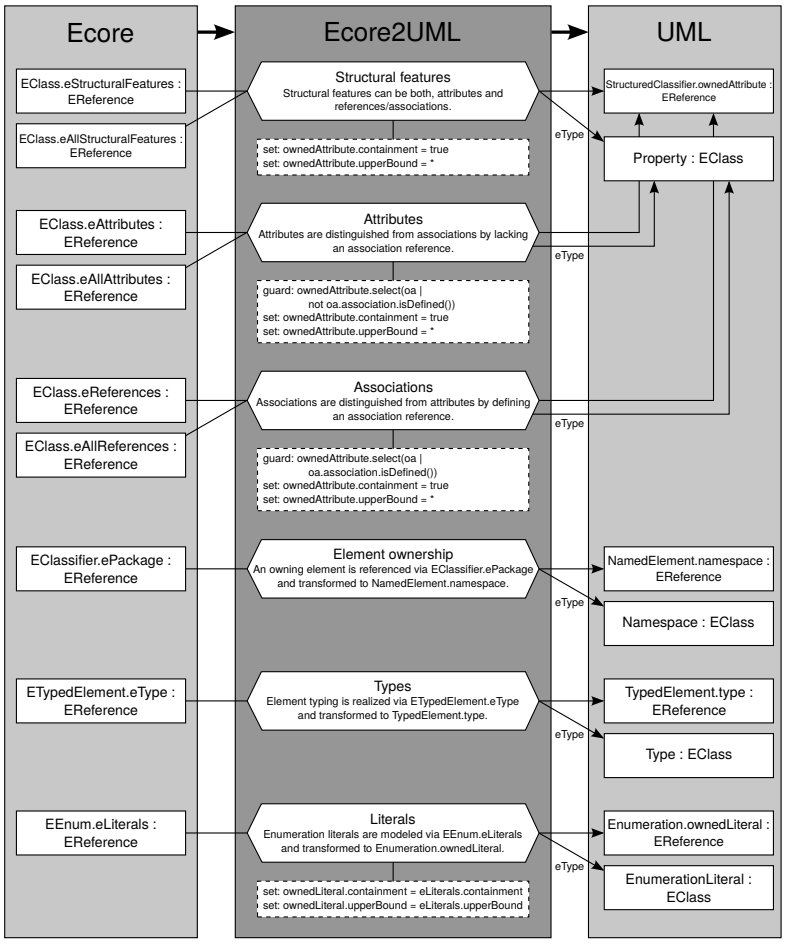

Figure 2: Mapping diagram of the Ecore2UML metamodel transformation operations.

In this migration scenario, the M2T transformation templates of Pongo should be refactored to work with models conforming to the UML metamodel. To test the refactored M2T templates, we converted the Ecore domain model of the Pongo-based blogging system to a UML class diagram. For this M2M transformation, we used the Ecore-to-UML conversion functionality provided in the Sample Ecore Model Editor of the Eclipse Modeling Framework (EMF).

3.3 Base Metrics

For the scope of this paper, a (software/quality) metric "is a term that embraces many activities, all of which involve some degree of software measurement" [FP97]; it is "a quantitative measure of the degree to which a system, component, or process possesses a given attribute" [Ins10]. According to the GQM model (see Table 1), measurement is performed on Pongo's M2T generator templates. For creating a benchmark data set, the standard-problem definition identifies the measurement constructs for quantifying the amount of refactorings on the generator templates applied/required to conform to the evolved (UML) metamodel. We adopted metrics from related work on model transformations, such as, for ATL (see, e.g., [KGBH10, vAvdB10, Vig09]):

1. *Number of automatically refactored concepts:* This metric counts every automatically adapted concept (e.g., types, expressions) required to render M2T transformations compliant with the evolved metamodel (see, e.g., [FP97]).
2. *Number of errors in generated artifacts:* This metric counts the number of defects in the generated platform-specific software artifacts (i.e. source code) produced by the evolved M2T transformations (see, e.g., [FP97]).
3. *Number of called helpers*: The M2T generator templates of the blogging system use helper operations for their implementation. This metric counts the number of helper operations (e.g., defined in generator templates) which are used (i.e. called at least once) during M2T transformation (see, e.g., [vAvdB10, Vig09]).

4. *Number of calls to helpers*: This metric sums up the total number of calls to helper operations during a M2T transformation (see, e.g., [vAvdB10, Vig09]).
5. *Transformation size*: This metric counts the SLOC of all M2T transformation definitions involved (see, e.g., [FP97, KGBH10]).
6. *Execution time:* This metric measures the average runtime of executing the entire evolution process of M2T generator templates. Used hardware and software details must be provided (see, e.g., [FP97]).

The six metrics directly relate to the four criteria stated in Section 2: completeness, correctness, complexity, and runtime performance. All six metrics quantify internal attributes of a M2T transformation and can be measured directly (see, e.g., [FP97, vA10]). Collecting the various metrics can typically be automated, for example, by leveraging built-in introspection and profiling mechanisms offered by the metamodeling platforms.

Metric 1. (*number of refactored concepts*; answering the corresponding question defined in Table 1) is an indicator for *completeness*: A high number of automatically rewritten concepts performed by a system would establish evidence for the advantage of fewer manual adaptations. Metric 2. measures the *correctness* of generated software artifacts (i.e. the number of errors in the source code). If the evolved source code is identical with the benchmark version (e.g., checked with a diff tool), zero defects can be assumed. The metrics 3.–5. yield insights into the adaptation *complexity* and are also operationalized as a means of control. The *numbers of called/calls to helpers* are metrics to check if a potentially high number of automatically rewritten concepts are not based on extensive use of (newly defined) helper operations. Additional helper operations would also increase the *transformation size* of the generator templates. It is evaluated whether and how much the transformation size of the rewritten generator templates differ when compared with their baseline versions. The last metric 6. quantifies the *execution time* of the adaptation process and provides *runtime performance* indicators for a specific hardware and software configuration.

3.4 Derived Metric: Benchmark Score

To ease comparison between a benchmarked technique (implementation) and its benchmark technique (implementation), we propose a derived metric: a *benchmark score* (see, e.g., [vdBHVP11]). The benchmark score is based on computing a weighted aggregate for three of the four criteria introduced in Section 2. We excluded the performance criterion due to its strong context bias (hardware and software specifications).

The benchmark score (BS) is computed as follows: $BS_S = S_{M1} - S_{M2} - \frac{1}{3} \times (\frac{S_{M3}}{RI_{M3}} + \frac{S_{M4}}{RI_{M4}} + \frac{S_{M5}}{RI_{M5}})$. BS refers to the benchmark score of the system under evaluation (S); $M\{1..5\}$ to the respective data of the metric of either the reference implementation (RI) or the evaluated system (S). The score is calculated by taking the number of automatically refactored concepts by a system as basis and subtracting the number of errors in generated artifacts produced by the system as well as the equally weighted ratios of number of called helper operations, calls to helper operations, and the size of the M2T transformations to the respective reference figures.

If missing another technique (implementation) as a point of reference, and to set a worst-case scenario as a benchmark, the benchmark score can be based on reference figures for an extreme setting: the completely *manual* refactoring of Pongo under the Ecore/UML migration scenario. To arrive at these reference figures, we manually implemented the necessary refactorings for the evolution of the blogging M2T transformations to be UML-compliant (see Section 3.2). The suggested benchmark score then takes these worst-case figures as the benchmark data and captures the relative distance of an actually benchmarked technique (implementation) to these reference data. The theoretical maximum score for the Pongo standard problem is *60*, because this is the total number of template elements in need of refactoring (S_{M1}) found for the Pongo migration scenario (see also Table 2). Practially, the maximum score will be below its upper bound of 60, because it is unlikely that one finds empty M2T transformations ($SLOC = 0$; metric 5.).

4 Benchmarking Example: Higher-Order Rewriting of M2T Templates

In [HSS13], we present an approach to rewrite M2T generator templates syntactically to reuse them for evolving metamodels. By considering M2T templates as first-class models and by reusing M2M transformation traces, we developed a rewriting approach based on higher-order model transformations (HOTs) for *transformation modifications* [TJF+09]. To demonstrate the feasibility of this rewriting technique, we provide a prototype implementation and an integration example based on the EMF project and the Epsilon language family. Hence, with our rewriting approach for M2T templates, we provide a solution to deal with structural mismatches between

different metamodels in an evolution scenario. Our current approach supports three syntactical higher-order rewriting operations (retyping, association retargeting, and property renaming [HSS13]).

We benchmark our approach according to the criteria defined in Section 3. For the evolutionary scenario of migrating from Ecore to UML class models, the example explores the refactoring process of the M2T transformation definitions (to collect data for the metrics 1. and 6.). The rewritten M2T templates are applied on the evolved UML-based domain model of the blogging system (for metrics 3.–5.) to compare the generated platform-specific software artifacts (i.e. Java classes) with the ones originally created from the Ecore domain model (for metric 2.) to evaluate the successful transformation according to the benchmark's criteria.

Table 2 shows the data collected for the first five metrics by conducting the benchmarking example. Metric 1. counts the number of automatically rewritten concept occurrences in the Pongo M2T templates. In essence, we could automate all rewriting operations. Our benchmarking example did not introduce any defects in the generated Java source files (metric 2.) as checked by a diff tool and by executing Pongo's test application (i.e. the example generated the identical set of Java files consisting of a total of 282 SLOC). The (3.) number of called helper operations and the (4.) number of calls to helper operations are identical as for the manual reference implementation (see Section 3.4). However, our automatic refactorings increased the (5.) size of the M2T transformation specifications to a total of 505 SLOC. With these figures, our software system arrives at a benchmark score of *58.975*.

As the development of our benchmarking example was influenced by our manual implementation, the data for the corresponding metrics are similar. The increase in the SLOC of the M2T transformations in our example (7.45% when compared to the manual adaptation; see metric 5. in Table 2) is caused by the formatting instructions implemented in the model-to-code part of the code/model round-tripping for M2T generator templates. A layout-preserving round-tripping functionality of our approach may be beneficial for reducing the transformation size of the M2T templates.

Table 2: Results of our manual and example implementations for the metrics defined as benchmark.

Metric	Manual adaptation	HOT-based rewriting [HSS13]
1. Number of (automatically) refactored concepts	60	60
2. Number of errors in generated artifacts	0	0
3. Number of called helpers	12	12
4. Number of calls to helpers	305	305
5. Transformation size (SLOC)	470	505

Additionally, Table 3 provides an overview of the average execution times (in milliseconds; ms) of the transformations performed during the example (metric 6.).[6] We employed the Epsilon profiling mechanism [Kol07] as well as Java's `System.nanoTime()` method for measuring the time needed to execute a particular transformation. We report the arithmetic mean of executing every transformation ten times.[7] The actual rewriting of the M2T transformation models (i.e. executing the rewriting operations) required 2520±32ms; that is ≈59% of the total execution time of 4263±42ms. The remaining time was needed to load models, perform metamodel migration (Ecore2UML M2M transformations) and so forth. Overall, the average execution time of the complete benchmarking example sums up to 6579±68ms (for further discussions on our example, we refer to [Hoi14]).

Table 3: Rounded average execution times of the benchmarking example.

Transformation	Average execution time (in ms)
M2T generator templates to M2T transformation models	1266±45
Rewriting M2T transformation models	4263±42
Rewritten M2T transformation models to M2T generator templates	1051±23
Total	6579±68

[6] Execution times are measured on the following hardware and with the following software specifications: Intel Core i5-3320M CPU 2.6 GHz, 12 GB RAM, 64-bit Ubuntu 13.04, Eclipse 4.2.

[7] The value after ± shows the standard deviation (σ) from the arithmetic mean ($\bar{x}$).

5 Discussion

The proposed standard problem is meant to evaluate evolving software systems which build on M2T generator templates. Note that, although we adopted a particular MDD technology stack (i.e. EMF, Epsilon) along selecting the application (Pongo), the fundamentals of the standard-problem definition do not rely on any of these tools. Critical artifacts such as metrics and metamodel transformations (Ecore2UML) are technology-agnostic. We demonstrated that the standard problem and benchmarking setup can host HOTs (as in our example) and a specific M2M transformation language (ETL [KRGDP15]). Likewise, it could use another support technique for coupled evolutions (e.g. adapter models) and an alternative model-transformation language (e.g. ATL).

However, any observations based on these metrics are specific to a given technique, implementation, and technology stack. Therefore, details of the benchmarking setup (e.g., the reference data on manual refactorings) must be calibrated for a targeted technology stack. Hence, to compare multiple approaches via the benchmark score—although defined in a generic manner—, each implementation must be based on the same or closely comparable MDD technologies.

Currently, one barrier to portability is that a technique (implementation) under evaluation must be able to handle the evolution of EGL-based M2T generator templates (as the dedicated Epsilon dialect to specify M2T transformations). Nevertheless, the standard-problem asset package is open for contributions of idempotent M2T transformation definitions expressed in alternative M2T languages (e.g., Acceleo/MOFM2T).

As for the metamodel-evolution scenario supported (migration), our benchmarking setup targets the well known and frequently adopted two metamodels: Ecore and UML. However, the design of our standard problem can be extended to include similar metamodel evolutions commonly considered (e.g., activity models from UML 1.* to UML 2.* in [RHM+14]). In addition, selected parts (e.g., base and derived metrics) can be reused and adapted to other evolution scenarios (e.g., replacement/rewrite, application improvement [UN10]).

Our benchmarking setup employs certain metrics in order to quantify characteristics of a M2T transformation system, thereby neglecting other quality aspects (e.g., ease of use, stability etc.). As our metrics are adopted from related work (see, e.g., [KGBH10, vAvdB10, Vig09]), we rely on the authors' demonstration that the metrics are acceptable for their intended use (in the context of model transformations). It is neither the goal of this paper to validate the adopted metrics for their representation condition, nor for their theoretical or empirical validity regarding a given attribute (for a discussion on validating software metrics, see, e.g., [MSW13]).

6 Related Work

In [LW13, WL13], the authors present benchmarks for types of model evolution systems. In particular, [LW13] proposes a benchmark for model versioning systems that support collaborative model-based development (e.g., for conflict detection when merging changes into a consolidated model version). The benchmark enables the automatic evaluation of conflict detection components. In contrast, [WL13] addresses the case of matching heterogeneous models that do not have a common predecessor. The proposed benchmark consists of real-world metamodels and manually defined expected correspondences that allow to evaluate automatically the quality of the output of model matching systems. Both benchmarks have in common that they employ a subset of the same evaluation criteria as we do (completeness, correctness, performance). However, the focus of the benchmarks is not on evaluating the evolution of M2T transformations but on model versioning and matching systems.

The authors of [WKK+10] present a feature-based classification of heterogeneities between object-oriented metamodels. This provided the basis for establishing benchmark examples that allow the evaluation of existing approaches with respect to their ability to resolve such heterogeneities. In our work, we reuse the classification from [WKK+10] to categorize structural heterogeneities between the Ecore and UML metamodels. The different benchmark examples share similarities with the Ecore2UML evolution scenario presented in this paper. However, the assessment of these examples are not guided by quantifiable measurements and, thus, the evaluation of a system largely depends on the interpretation of the respective evaluator. Nevertheless, the authors of [WKK+10] provide a source for extending our definition of a standard problem to cover additional heterogeneity aspects. In this sense, their work can be seen as complementary.

In [vdBHVP11], the authors present a method for assessing the quality of model comparison systems as well as a data set to be used for controlled evaluation experiments. Together with the results obtained by assessing two software systems, the authors constitute a benchmark for model comparison systems. Although sharing some evaluation criteria (completeness, correctness, performance), the metrics focus on model differences on the level of model elements (e.g., deleted, added elements)—represented in an instance model of a dedicated difference metamodel. Benchmarking the evolution of M2T generator templates is not covered in [vdBHVP11].

The authors of [vABKFP11] investigate the factors that have an impact on the execution performance of model transformations. In particular, the performance of three model transformation languages are analyzed (ATL, QVT operational mappings, QVT relations). In our work, we adopt two of the metrics (number of called helpers, number of calls to helpers) proposed by the same author in an earlier publication [vAvdB10]. Although van Amstel et al. focus specifically on evaluating performance criteria for the ATL and QVT M2M transformation languages, some of their proposed metrics can be generalized (as we did for this paper). However, their objective is not to design a benchmark for model transformation systems.

7 Conclusion and Future Work

In this paper, we report on the foundations of a standard-problem definition for systematically comparing coupled evolution in M2T transformation systems. The standard problem provides a basis for the analytical and quantitative comparison (benchmarking) of different techniques (implementations) in support co-evolving M2T generator templates: HOTs, generic templates, and adapter models. Benchmarking reflects completeness, correctness, complexity, and performance of M2T transformation adaptations in terms of a benchmark score. To validate the approach tentatively, we benchmarked our approach of higher-order rewriting M2T templates.

As next steps, and based on the feedback of the AMT community, we will extend the standard problem to include more challenging kinds of metamodel heterogeneity [WKK+10]. For this, we will also assess additional metrics for inclusion (e.g., similarity of relations [KGBH10]). We believe that the Ecore2UML migration scenario underlying the problem definition is a promising candidate of a future standard problem (similar to the one successfully defined for software product lines [LHB01]). It benefits from simplicity and understandability while relying on critical building blocks for a whole MDD ecosystem. However, this paper presents only a first step towards the definition of a comprehensive standard problem.

References

[CP10] W. Cazzola and D. Poletti. DSL evolution through composition. In *Proc. 7th Worksh. Reflection, AOP and Meta-Data for Softw. Evol.*, pages 6:1–6:6. ACM, 2010.

[FP97] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach.* PWS Publishing Company, 2nd edition, 1997.

[GdLK+13] E. Guerra, J. de Lara, D. Kolovos, R. Paige, and O. dos Santos. Engineering model transformations with transML. *Softw. Syst. Model.*, 12(3):555–577, 2013.

[Gen10] Generative Software. Umfrage zu Verbreitung und Einsatz modellgetriebener Softwareentwicklung. Survey Report, Generative Software GmbH and FZI Forschungszentrum Informatik, 2010. Available at: `http://www.mdsd-umfrage.de/mdsd-report-2010.pdf`.

[Hoi14] B. Hoisl. *Integration and Test of MOF/UML-based Domain-specific Modeling Languages.* PhD thesis, WU Vienna University of Economics and Business, November 2014.

[HSS12] B. Hoisl, M. Strembeck, and S. Sobernig. Towards a systematic integration of MOF/UML-based domain-specific modeling languages. In *Proc. 16th IASTED Int. Conf. Softw. Eng. and Appl.*, pages 337–344. ACTA Press, 2012.

[HSS13] B. Hoisl, S. Sobernig, and M. Strembeck. Higher-order rewriting of model-to-text templates for integrating domain-specific modeling languages. In *Proc. 1st Int. Conf. Model-Driven Eng. and Softw. Dev.*, pages 49–61. SciTePress, 2013.

[HSS14] B. Hoisl, S. Sobernig, and M. Strembeck. Natural-language scenario descriptions for testing core language models of domain-specific languages. In *Proc. 2nd Int. Conf. Model-Driven Eng. and Softw. Dev.*, pages 356–367. SciTePress, 2014.

[Ins10] Institute of Electrical and Electronics Engineers (IEEE). Systems and software engineering – vocabulary. Available at: `http://standards.ieee.org/findstds/standard/24765-2010.html`, 2010. ISO/IEC/IEEE 24765:2010.

[KGBH10] L. Kapová, T. Goldschmidt, S. Becker, and J. Henss. Evaluating maintainability with code metrics for model-to-model transformations. In *Research into Practice – Reality and Gaps*, volume 6093 of *LNCS*, pages 151–166. Springer, 2010.

[Kol07] D. S. Kolovos. An overview of the Epsilon profiling tools. Technical Report version 1.3, The University of York, July 2007. Available at: `https://www.eclipse.org/epsilon/doc/EpsilonProfilingTools.pdf`.

[Kol15] D. Kolovos. Pongo: 5 minute tutorial. Available at: https://code.google.com/p/pongo/wiki/5MinuteTutorial, 2015.

[KRGDP15] D. Kolovos, L. Rose, A. García-Domínguez, and R. Paige. The Epsilon book. Available at: http://www.eclipse.org/epsilon/doc/book/, 2015.

[KW15] D. Kolovos and J. R. Williams. Pongo: Java POJO generator for MongoDB. Available at: https://code.google.com/p/pongo/, 2015.

[LHB01] R. E. Lopez-Herrejon and D. S. Batory. A standard problem for evaluating product-line methodologies. In *Proc. 3rd Int. Conf. Genera. Compon.-Based Softw. Eng.*, pages 10–24. Springer, 2001.

[LW13] P. Langer and M. Wimmer. A benchmark for conflict detection components of model versioning systems. *Softwaretechnik-Trends*, 33(2), 2013.

[MSW13] A. Meneely, B. Smith, and L. Williams. Validating software metrics: A spectrum of philosophies. *ACM T. Softw. Eng. Meth.*, 21(4):24:1–24:28, February 2013.

[RHM+14] L. M. Rose, M. Herrmannsdoerfer, S. Mazanek, P. V. Gorp, S. Buchwald, T. Horn, E. Kalnina, A. Koch, K. Lano, B. Schätz, and M. Wimmer. Graph and model transformation tools for model migration. *Softw. Syst. Model.*, 13(1):323–359, 2014.

[RIP12] D. D. Ruscio, L. Iovino, and A. Pierantonio. Coupled evolution in model-driven engineering. *IEEE Softw.*, 29(6):78–84, 2012.

[RRIP14] J. D. Rocco, D. D. Ruscio, L. Iovino, and A. Pierantonio. Dealing with the coupled evolution of metamodels and model-to-text transformations. In *Proc. Worksh. Models and Evol.*, volume 1331, pages 22–31. CEUR Worksh. Proc., 2014.

[SHSed] S. Sobernig, B. Hoisl, and M. Strembeck. Extracting reusable design decisions in UML-based domain-specific languages: A multi-method study. Submitted.

[SV06] T. Stahl and M. Völter. *Model-Driven Software Development.* John Wiley & Sons, 2006.

[SWCD12] G. M. Selim, S. Wang, J. R. Cordy, and J. Dingel. Model transformations for migrating legacy models: An industrial case study. In *Proc. 8th Europ. Conf. Model. Found. Appl.*, volume 7349 of *LNCS*, pages 90–101. Springer, 2012.

[TJF+09] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the use of higher-order model transformations. In *Model Driven Archit. – Found. and Appl.*, volume 5562 of *LNCS*, pages 18–33. Springer, 2009.

[UN10] W. H. Ulrich and P. H. Newcomb. *Information Systems Transformation: Architecture-Driven Modernization Case Studies.* Morgan & Kaufmann Publishers, 2010.

[vA10] M. van Amstel. The right tool for the right job: Assessing model transformation quality. In *Proc. 34th Annu. IEEE Comput. Softw. and Appl. Conf. Worksh.*, pages 69–74, July 2010.

[vABKFP11] M. van Amstel, S. Bosems, I. Kurtev, and L. Ferreira Pires. Performance in model transformations: Experiments with ATL and QVT. In *Theory and Practice of Model Tran.*, volume 6707 of *LNCS*, pages 198–212. Springer, 2011.

[vAvdB10] M. van Amstel and M. van den Brand. Quality assessment of ATL model transformations using metrics. In *Proc. 2nd Int. Worksh. Model Tran. with ATL*, volume 711, pages 19–33. CEUR Worksh. Proc., 2010.

[vdBHVP11] M. van den Brand, A. Hofkamp, T. Verhoeff, and Z. Protić. Assessing the quality of model-comparison tools: A method and a benchmark data set. In *Proc. 2nd Int. Worksh. Model Comparison in Practice*, pages 2–11. ACM, 2011.

[Vig09] A. Vignaga. Metrics for measuring ATL model transformations. Technical Report TR_DCC-20090430-006, Universidad de Chile, April 2009. Available at: http://swp.dcc.uchile.cl/TR/2009/TR_DCC-20090430-006.pdf.

[vSB99] R. van Solingen and E. Berghout. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development.* McGraw-Hill, 1999.

[WKK+10] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Towards an expressivity benchmark for mappings based on a systematic classification of heterogeneities. In *Proc. 1st Int. Worksh. Model-Driven Interoperability*, pages 32–41. ACM, 2010.

[WKS+10] M. Wimmer, A. Kusel, J. Schönböck, W. Retschitzegger, W. Schwinger, and G. Kappel. On using inplace transformations for model co-evolution. In *Proc. 2nd Int. Worksh. Model Tran. with ATL*, volume 711, pages 65–78. CEUR Worksh. Proc., 2010.

[WL13] M. Wimmer and P. Langer. A benchmark for model matching systems: The heterogeneous metamodel case. *Softwaretechnik-Trends*, 33(2), 2013.