A Generic Framework for Model-Set Selection for the Unification of Testing and Learning MDE Tasks

Edouard Batot
DIRO, Université de Montréal, Canada
batotedo@iro.umontreal.ca

Houari Sahraoui
DIRO, Université de Montréal, Canada
sahraouh@iro.umontreal.ca

ABSTRACT

We propose a generic framework for model-set selection for learning or testing Model-Driven Engineering tasks. We target specifically tasks that apply to or manipulate models, such as model definition, model well-formedness checking, and model transformation. In our framework, we view the model-set selection as a multi-objective optimization problem. The framework can be tailored to the learning or testing of a specific task by firstly expressing the coverage criterion, which will be encoded as a first optimization objective. The coverage is expressed by tagging the subset of the input metamodel that is relevant to the considered task. Then, one or more minimality criteria are selected as additional optimization objectives. We illustrate the use of our framework with the testing of metamodels. This case study shows that the multi-objective approach gives better results than random and mono-objective selections.

1. INTRODUCTION

Model-Driven Engineering (MDE) aims at raising the level of abstraction in software development. It promotes the use of domain-specific languages (DSLs) that can help domain experts to model their applications and rely on automation for development and maintenance tasks, such as model transformation, code generation, and model refactoring [1]. Unlike for the traditional software development, development artifacts and tasks in MDE (metamodels, transformations, etc.) are themselves domain dependent and require specific knowledge for their definition and testing. Thus, it is important to ensure that they are well defined.

When the domain knowledge is available and explicit enough, it can be used to manually define the development artifacts such as modeling languages and transformations. These are then tested to ensure their correctness. However, when this knowledge is incomplete or difficult to explicit, existing research has shown us that MDE development artifacts can be learned from examples [2, 3, 4, 5, 6, 7]. In both cases, learning and testing, having representative sets examples is crucial condition of success.

As models are the core concepts of MDE, it is natural that most of the development artifacts apply to or manipulate models. For example, well-formedness rules check if a given model is valid with respect to the domain knowledge. Similarly, model transformations, refactoring rules, and other activities such as model merging and differentiating, also take models as inputs. Consequently, the examples for learning and testing such artifacts/tasks are sets of models together with their respective task outputs/oracles. For instance, a learning example or a test case for metamodel well-formedness rules is a model with, as output/oracle, its actual validity. For transformations, transformation rules/programs can be learned from or tested with a representative set of input models and their respective expected output models. Whereas input model sets can be automatically generated, the tasks' outputs/oracles must be provided by the expert.

Much work has been conducted on generating and selecting models and model sets for learning and testing MDE development tasks, see, for example, [8, 4] for metamodel testing, [9, 10, 11, 12] for transformation testing, [13] for metamodel definition, etc. However, all this contributions target a specific task (metamodeling, transformations, etc.) with a specific purpose (manual definition, automated learning, or testing). Consequently, they are difficult to generalize and reuse. In this paper, we propose a generic framework, our approach, for model-set selection for learning or testing Model-Driven Engineering tasks. We target specifically tasks that apply to or manipulate models, such as modeling, model well-formedness checking, and model transformation. In our framework, we view the model-set selection as a multi-objective optimization problem. The framework can be tailored to the learning or testing of a specific task by firstly expressing the coverage criterion, which will be encoded as a first optimization objective. The coverage is expressed by tagging the subset of the input metamodel that is relevant to the considered task. Then, one or more minimality criteria are selected as additional optimization objectives.

To illustrate the use of our approach, we address the well-known problem of metamodel testing [8]. We evaluate our approach on this case by comparing the generated model sets with those obtained by mono-objective and random strategies. The results of this study show that the multi-objective approach gives better results than the other strategies and that different combinations of the coverage with the minimality objectives lead to different trade-offs in terms of coverage and size of the model sets.

The remainder of the paper is organized as follows. In Section 2, we present the basic definitions and illustrate the commonalities and differences when learning/testing MDE tasks. Section 3, describes our approach, the unified model-set selection framework. We illustrate the use our approach in Section 4 with the metamodel testing case. Section 4.1 presents the setup and the results of an empirical study. Finally, after discussing the related work in Section 6, we give a discussion and a conclusion in Section 7.

MODELS '16, October 02 - 07, 2016, Saint-Malo, France

ISBN 978-1-4503-4321-3/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976767.2976785

2. PROBLEM STATEMENT

The problem we solve in this paper is the definition of a generic framework for the selection of model sets that can be used to define, learn or test MDE tasks. In MDE, models are the main concepts. It is then normal that most of the tasks use or manipulate one or more models. In the remainder of this section, we present examples of tasks manipulating models, and explore their commonalities. Then, we propose a common terminology, which will be used in the remainder of the paper.

2.1 Task Examples

2.1.1 Metamodeling

Metamodeling is basically the definition of a modeling language, since it provides a way of describing the whole class of models that can be represented by that language. This task needs model examples to ensure, when defining the metamodel, that the concepts brought by the domain expert are properly considered [14]. An example for a metamodeling task is composed of a model, which conform to the already-defined version of the metamodel, and its validity with respect to the represented domain. Thus, after defining a version of the metamodel, one can generate a set of models that covers as much as possible the modeling space defined by the current version [13]. Although a model set can be automatically generated, the validity of the models to the represented domain must be manually provided by an oracle/expert. As this manual operation can be costly, it is better that the generated set of models is as minimal as possible.

2.1.2 Model Well-Formedness Checking

After defining a metamodel, a next step is to reduce the metamodel scope to avoid ill-formed configurations/instances [14]. This is done by producing well-formedness rules (WFR), which qualify models being or not part of an application domain. WFR can be defined manually when the knowledge about the domain is complete and explicit enough to be formalized in, for instance, OCL expressions. In that case, model examples are used to test these rules. Alternatively, WFR can be learned from examples [5]. In this work, WFR are learned from a set of examples and counter-examples. For learning and testing WFR, examples are model instances tagged as valid whereas counter-examples are model instances tagged as invalid. Like for the metamodeling task, generating model instances can be done automatically. However, tagging them as valid/invalid is a manual task that cannot be performed on large sets of models in a reasonable timeframe.

2.1.3 Model Transformation

A model transformation (MT) is the process of transforming a model into another model or a textual representation according to a transformation specification. MTs are actually defined at the metamodel level, and then applied at the model level on models that conform to the considered metamodels [15]. When a transformation program is defined, it can be tested by verifying that this program produces the expected outputs for a given set of input models [16], or more generally, outputs that conform to a given oracle [17]. A transformation program can be learned from a set of *input models* and their corresponding output model [2, 18]. Hence, a case (example) for testing (learning) a model transformation is composed of an *input model* and, for instance, its associated expected output model [19]. The quality of a learned transformation and the accuracy of testing a transformation both rely on the representativeness of the *input models* with respect to the input space of that transformation, described by its input metamodel. Actually, a transformation may concern only part of the input metamodel as stated in [20] and [11]. As a consequence, automatic generation of models to learn/test MT should maximize the coverage of the metamodel part concerned by the transformation while minimizing the number and size of the generated models. As for the two previous tasks, the output part of the *cases* is expected to be manually specified/provided. The same reasoning holds for model refactoring, which is a particular kind of transformation. In this case, the refactoring operations apply to instances of only a subset of metamodel concepts.

2.2 Commonalities and Terminology

In addition to the previously-mentioned examples, many tasks can be tested and/or learned by means of selected *cases*, with each *case* having an *input model* (sometimes more than one) and a manually specified/given output. Examples of such activities are model evolution, code-generation, and model merge, to name a few.

We showed in this section that these MDE tasks can be tested following a same testing scenario. They take a model as input for their test cases, and require an expert to write the expected output [21]. We showed too that learning an automated task in MDE requires the same data as the testing: *input models* and their corresponding expected task outputs. Moreover, for all tasks, *input models* of the *cases* must cover as much as possible an input space defined by the input metamodel. The coverage definition varies from one task to another by determining which parts of the input metamodel are concerned by the task. Finally, another common part to these activities is that providing the oracle for each generated *input model* is time and effort consuming. Hence, a model set selection needs to consider the number/size of the retained models.

To use a uniform vocabulary for the rest of this paper, we introduce in the following paragraphs a common terminology. We start by defining the concept of ***task***.

DEFINITION 1. *A **task** is any* MDE *activity that applies to or manipulates models.*

A task can be manual such as defining the structure of a metamodel. In that situation, giving model instances to the metamodeler helps ensuring that the defined version is correct with respect to the domain knowledge. The task can be automatic such in the case of model transformation, refactoring, or validity checking with well-formedness rules. The second important concept to define is the ***model-set selection***.

DEFINITION 2. *The **model-set selection** is the process of selecting, from a model base, a set of models that satisfies one or more selection objectives.*

The model base can be gathered from existing material or randomly generated. Another important concept to define is the ***purpose*** of the model-set selection.

DEFINITION 3. *The **model-set selection purpose** is the step in the lifecycle of the task for which the selected models are necessary. A step can be the manual definition, the automated learning or the testing of the targeted task.*

Finally, the last concept we define is the ***case***.

DEFINITION 4. *Depending on the purpose and the task for a model set selection, a **case** is a pair composed of (1) a model in the selected set and (2) the expected task output corresponding to this model. A **case** can be a test case or a learning example.*

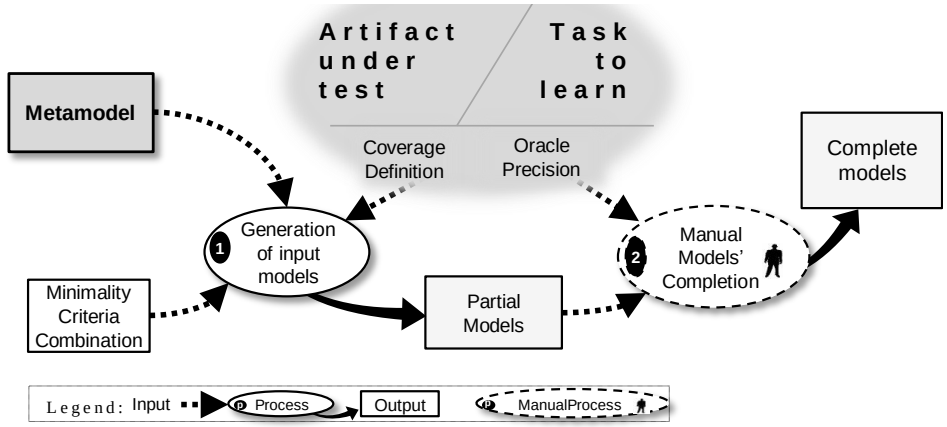

Figure 1: A generic framework for testing MDE activities

3. FRAMEWORK

As stated in Section 2, MDE tasks diverge on their coverage definition and in the type of expected outputs. Fig. 1 presents the general architecture of our unified framework. A metamodel, together with a coverage definition, *i.e.,* its subset, provide sufficient information to select a set of representative *input models*. A choice between different minimality criteria, which can be combined together, is offered to take into account variations in the minimality definition. Then, each selected model needs to be completed by an expert to specify the output expected after the execution of the task on this *input model*. Coverage definition and output specification are highly task dependent. In this paper, we focus on the first step, *i.e.,* the selection of *input models*. In the remainder of this section, we present the criteria to optimize during the model set selection. Then, we explain how the framework combines them in order to select model sets as a multi-objective optimization problem.

3.1 Model set Selection Objectives

3.1.1 Coverage Definition

As we are dealing with *inputs models*, the coverage that must be satisfied by the selection process can be expressed in terms of the models' metamodel. Still, different tasks may require different modeling spaces. As stated in Section 2, a model transformation task may be concerned only by the subset of the metamodel's elements that are targeted by the transformation, whereas, testing a metamodel may involve the whole metamodel. In our framework, the coverage criterion is expressed as a subset of the metamodel elements, which are tagged as mandatory. The remaining elements are optional in the sense that the selected models may include instances of them, but these are not considered when evaluating the coverage.

Choosing and applying a coverage definition is an open topic which usually requires knowledge from the application domain. For some tasks, the involved metamodel elements can be enumerated explicitly. For other, the set of metamodel's elements must be deduced from a high level specification. To illustrate this second situation, let us consider the task of learning well-formedness rules.

Suppose we want to learn WFRs to complete the definition of the Ecore metamodel (see the partial illustration in Fig. 2). An intuitive approach to define the coverage is to consider the whole Ecore metamodel. However, Cadavid, after studying dozens of metamodels, shows in [22] that there are 21 OCL patterns that are

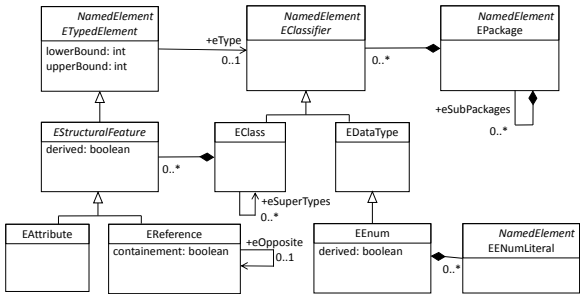

Figure 2: Ecore metamodel

used to express all the WFRs independently from the considered metamodel. Of course, these patterns are expressed at the meta-meta-level (MOF). Consequently, for learning WFRs, the process, rather than searching for any OCL expression that can apply to Ecore, it searches only for OCL expressions that are instances of these patterns in this metamodel [5].

A pattern definition contains the *MOF structure* involved, an *OCL Expression Template*, and parameters. The *MOF structure* characterizes the structural situations in which the pattern may apply (*e.g.,* classes and features involved). The *OCL Expression Template* defines the type of WFRs by explaining how the listed parameters are used to express these rules. The following description details the example of *AcyclicReference* pattern.

- **MOF Structure:** The pattern applies whenever there is a class containing a reference which type is itself. Also, the upper bound of this reference has to be "many"; this is because the OCL expression invokes on this attribute the operation closure, which can only be invoked on collections.

- **OCL Expression Template:**
  context ClassA **inv** AcyclicReference : attributeA
  ->closure(iterator: ClassA | iterator.attributeA)
  ->excludes(self) ;

- **Parameters:** ClassA, AttributeA

If we want to learn WFRs of type *AcyclicReference* in Ecore, we have to generate model cases that cover the different instances of this pattern in Ecore. As shown in Fig. 4, classes "EClass" and

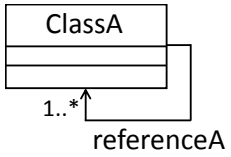

Figure 3: MOF structure of pattern *AcyclicReference*

"EPackage" with their corresponding one-to-many references, respectively "superTypes" and "subPackages" are the only instances of this pattern.

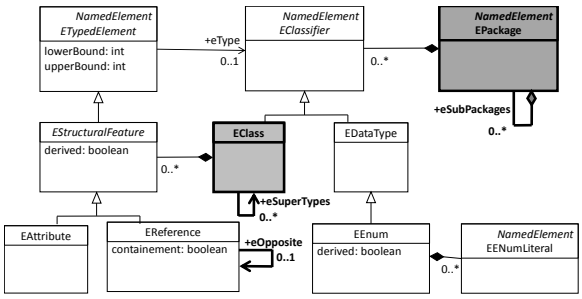

Figure 4: Elements of Ecore tagged as mandatory w.r.t. the pattern AcyclicReference

When considering more patterns, other elements, corresponding to instances of these patterns, are then added in the tagged list. For example, consider the pattern, *ReferenceDifferentFromSelf*, which involves classes with cyclic references with 0..1 cardinalities. Then, a match is the class "EReference" and its reference "eOpposite". These are also tagged as mandatory elements as depicted in Fig. 5. The same reasoning holds for model refactoring learning or testing. Here the elements to tag are deduced from generic refactoring operations.

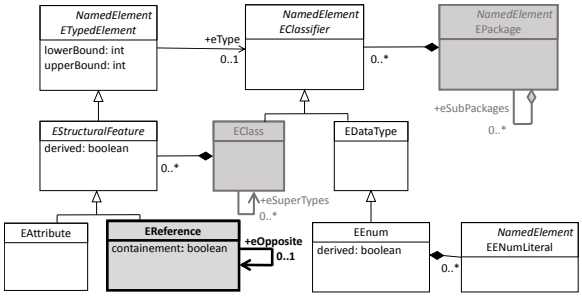

Figure 5: Elements of Ecore tagged as mandatory w.r.t. the patterns AcyclicReference and ReferenceDifferentFromSelf

Once the mandatory part of the metamodel is defined according to the considered task and purpose, the coverage calculation is generic and metamodel-independent in our framework as we will explain it in Section 3.2.2.

3.1.2 Minimality criterion

Selection of a set of models with a high coverage could be easily reached if we were not limited on the number and size of the selected models. However, recall that for each selected model, we have to complete manually the case by providing the task output for this model. For example, to test (or learn) a transformation, an expert could be asked to produce manually a transformed model for each selected model.

For this reason, the coverage objective should be mitigated by a minimality objective to find a good tradeoff between the model set coverage and its size. This idea of combining both objectives in MDE is not new. In [4], Cadavid *et al.* combines the coverage and a dissimilarity criterion in a single objective function to guide the model generation process. As the selected models should be as dissimilar as possible, solutions with a lot of models are expected to be penalized by this objective. A limitation to this dissimilarity criterion is that it is pair-wise, and that two models cannot be fully dissimilar as recognized by the authors. Thus, they tolerate a certain overlap ratio, given as a parameter. Depending on the value of this parameter, the number of selected models may be very large.

In our framework, in addition to the dissimilarity between models in a set (DIS), we consider two other minimality criteria. Firstly, we aim at minimizing the size of the model set in terms of the number of models (MIN). This criterion, considered alone, could results in a single (or a few) very large model(s) with a good coverage. To circumvent this side effect, we can consider the size of the model set in terms of the total number of elements contained in all the models of the set (MIN-R). In summary:

- **DIS** guarantees models to be dissimilar pair-wise in the solution set.
- **MIN** guarantees the model set to be kept as small as possible in terms of the number of models.
- **MIN-R** guarantees the models, in the solution set, to be as small as possible in terms of instantiated objects.

Each minimality objective has its own advantages and drawbacks. When using our framework, in addition to the coverage objective, one can select one or more minimality criteria depending on the targeted task. However, although selecting many minimality objectives may take advantages from them, it is known that increasing the number of objectives may compromise the chances to converge towards the optimal solution. The implementation of these objectives into fitness functions is detailed in Section 3.2.2. The evaluation of their combinations is studied in 4.1.

3.2 Model-set Selection as MOOP

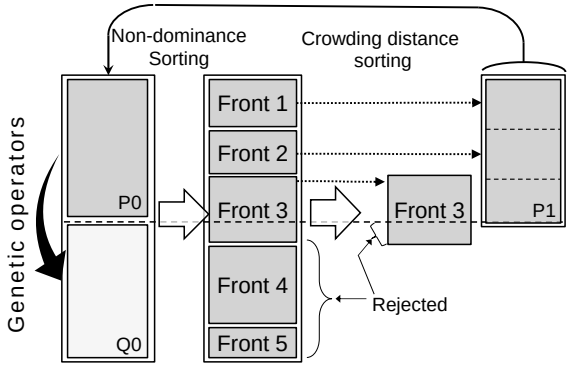

Figure 6: NSGA-II [23]

Coverage and minimality objectives being conflicting in essence, we represent the model set selection as a multi-objective optimization problem, and we solve it using the non-sorting genetic algorithm NSGA-II [23]. Before explaining how we adapt this algorithm to our problem, let us introduce some basic definitions.

DEFINITION 5. *A **multi-objective optimization problem** (MOOP) consists in minimizing or maximizing an objective function vector $f(x) = [f_1(x), f_2(x), ..., f_M(x)]$ of M objectives under some constraints. The set of feasible solutions,* i.e., *those that satisfy the problem constraints, defines the search space Ω. The resolution of a MOOP consists in approximating the whole **Pareto front**.*

DEFINITION 6. ***Pareto optimality:** In the case of a minimization problem, a solution $x^* \in \Omega$ is Pareto optimal if $\forall x \in \Omega$ and $\forall m \in I = \{1, ..., M\}$ either $f_m(x) = f_m(x^*)$ or there is at least one $m \in I$ such that $f_m(x) > f_m(x^*)$. In other words, x^* is Pareto optimal if no feasible solution exists, which would improve some objective without causing a simultaneous worsening in at least another one.*

DEFINITION 7. ***Pareto dominance:** A solution u is said to dominate another solution v (denoted by $f(u) \preceq f(v)$) if and only if f(u) is partially less than f(v),* i.e., *$\forall m \in \{1, ...M\}$ we have $f_m(u) \leq f_m(v)$ and $\exists m \in \{1, ..., M\}$ where $f_m(u) < f_m(v)$.*

DEFINITION 8. ***Pareto optimal set:** For a MOOP f(x), the Pareto optimal set is $P^* = \{x \in \Omega | \neg \exists x' \in \Omega, f(x') \preceq f(x)\}$.*

The idea of NSGA-II [23] is to make a population of candidate solutions evolve toward the near-optimal solution in order to solve a multi-objective optimization problem. NSGA-II is designed to find a set of optimal solutions, called non-dominated solutions, also Pareto set. A non-dominated solution is the one which provides a suitable compromise between all objectives without degrading any of them. As described in Fig. 6, the first step in NSGA-II is to create randomly a population P_0 of $N/2$ individuals encoded using a specific representation. Then, a child population Q_0, of the same size, is generated from the population of parents P_0 using genetic operators such as crossover and mutation. Both populations are merged into an initial population R_0 of size N, which is sorted into dominance fronts according to the dominance principle. The first (Pareto) front includes the non-dominated solutions; the second front contains the solutions that are dominated only by the solutions of the first front, and so on and so forth. The fronts are included in the parent population P_1 of the next generation following the dominance order until the size of $N/2$ is reached. If this size coincides with part of a front, the solutions inside this front are sorted, to complete the population, according to a crowding distance which favors diversity in the solutions [23]. This process will be repeated until a stop criterion is reached, *e.g.,* a number of iterations or all objectives greater than .99.

3.2.1 Solution Representation and Solution Creation

As our goal is to maximize the coverage of a metamodel or a subset of it, with a minimal set of models, a solution for our problem refers simply to a set of models. Actually, we transform the model generation problem into a model selection one. In a first step, we randomly generate a base of models for the considered metamodel. To this end, we use AtlanMod instantiator [1]. This tool allows to generate models in XMI format from a metamodel described in ECORE. The expected number of models, number of objects per model, maximum number of attributes and references, and maximum depth of the references can be specified. The generator produces the required number of correct instances (invalid instances, w.r.t multiplicity constraints, identified by the tool are ignored). It is configured with a uniform distribution, *i.e.,* when a maximum number is given for attributes/references/depth, any number below the maximum has the same probability to occur. We repeat the generation process with different model sizes to produce a large base of examples. For our experiments, we generate 10000 models of 2 to 200 objects.

In a second step, our optimization algorithm explores the set of subset of the model base to select the one which satisfies the coverage and minimality objectives.

3.2.2 Objective Functions

The objective functions assess the ability of a solution to solve the problem under consideration. To evaluate solutions (*i.e.,* model sets), we consider the coverage and minimality objectives: maximizing the coverage of the problem space by the model cases, and constraining the resulting set to be as small as possible.

Coverage computation. To assess the coverage of the modeling space by a set of models, we use the work by Fleurey *et al.* [10]. This work is considered as the state-of-the-art in partitioning a metamodel in order to distillate interesting structures from its features. We rate coverage upon this assumption. Based on Ostrand *et al.* [24] category-partitioning method, the authors decompose the static structure of a metamodel in three hierarchical levels: the metamodel fragment partition MFP contains model fragments, themselves composed of object fragments.

An *object fragment* is the association of a possible value (or a range of values) to a structural feature (attribute or reference) in the metamodel. To this end, each feature is partitioned, beforehand, into a set of (ranges of) values. For example, an integer-type attribute P of a class C is partitioned into three categories: {P=0, P=1, P>1}. For a given metamodel, an object fragment is defined for each category of the partition of each attribute/reference, e.g., $of(P, 0)$.

A model fragment contains one or more object fragments. We considered two strategies to define the model fragments. For the *AllRanges* strategy, a model fragment is defined for each object fragment, e.g., $mf((P, 0))$. For the *AllPartitions* strategy, we define a model fragment for each attribute/reference as a set of object fragments of the corresponding partition, *e.g.,* $mf((P, 0), (P, 1), (P, > 1))$. Fleurey *et al.* present two more strategies in order to cope with the combinatorial explosion of the partitioning of large metamodels.

For a given model instance m_i, a model fragment mf_j is covered by m_i if all the object fragments in mf_j appear in m_i. We denote this property by $covering(mf_j, m_i) = true$. Starting from this, it is possible to derive the set of model fragments covered $MFC(ms)$ by a set of models ms (a solution in our problem). Then, our coverage objective function is defined as the proportion of model fragments covered by the candidate solution ms over the metamodel fragment partition MFP. Formally:

$$coverage(ms) = \frac{|MFC(ms)|}{|MFP|}$$

Recall that only the elements of the metamodel tagged as mandatory are considered, in the definition of MFP, for the coverage evaluation.

Minimality computation. We consider three different minimality criteria DIS, MIN, and MIN-R. For the dissimilarity DIS of a solution, we use the definition from Cadavid *et al.*in [4], formally:

$$DIS = 1 - \frac{excessCovering}{(MFRT \times \#fragmentsCovered)} \quad (1)$$

where $excessCovering$ is the number of fragments covered in more than one model of the solution, $\#fragmentsCovered$ is

[1]https://github.com/atlanmod/mondo-atlzoo-benchmark/tree/master/fr.inria.atlanmod.instantiator

the number of fragments covered by the model set, and

$$MFRT = \#fragments \times overlapRatio \quad (2)$$

The coefficient $overlapRatio$ refers to the tolerated percentage of overlap between two models. This is set to 0.1 in our evaluation, as suggested by Cadavid *et al.* [4].

Then, we consider as minimality MIN of a solution ms the number of its models. However, it might be interesting to normalize this objective in the interval $[0, 1]$. In this context, the worst case being that each model covers only one fragment of the partitioned metamodel, MIN is normalized by the number of fragments in the MFP. MIN is defined as follows:

$$MIN = 1 - \frac{|ms|}{|MFP|} \quad (3)$$

Finally, to take into consideration the variation in size of models among the solution set, the third minimality criteria MIN-R uses the number of objects instantiated in the models of the solutions. Here again, we normalize the size by the number of fragments and the average size of the set models. Formally,

$$MIN_R = 1 - \frac{\sum_{m_i \in ms}(size(m_i))}{|MFP| \times avg\_model\_size} \quad (4)$$

where $size(m_i)$ is the size in term of objects of a model m_i and avg_model_size is the average size of the solution's models.

For MIN and MIN-R, our normalization uses approximations of the maximum number of models and the maximum number of objects in a solution. These numbers can be under-estimated, which could results in negative values for MIN and MIN-R. In that situations, we set the lower bound of MIN and MIN-R to 0. The goal of normalizing the three minimality criteria is to allow the combination of all the objective values, including the coverage, when selecting a unique solution among the Pareto front as explained in Section 4.1.

3.2.3 Genetic Operators

As mentioned earlier in this section, when evolving a population of solutions,NSGA-II derives new solutions from existing ones using crossover and mutation operators. The goal of the crossover is to find new, and possibly better, combinations of the genetic material present in a population. The mutation allows to inject new genetic material to possibly improve the population of solutions. As illustrated in Fig. 7, the *crossover* operator, in our framework, uses the single cut-point crossover. Each parent solution (set of models) is divided into two model subsets according to a randomly picked cut point. Then the model subsets of the parent solutions are exchanged to form two new model sets.

The *mutation* operator selects randomly a model in a solution (model set) and replaces it by a new model randomly picked in the model base.

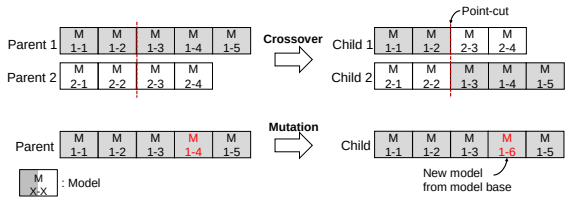

Figure 7: Crossover and mutation adapted in our framework

4. CASE STUDY: METAMODEL TESTING

MDE considers metamodels as development artifacts. They can be used to derive other artifacts automatically. We need, then, to ensure that a metamodel is correct before using it. Papers focusing on metamodel testing remain scarce. Wu *et al.* [25], in their literature review, highlight four main intents in metamodel instance generation/selection, and testing is not part of them.

During the metamodeling activity, the first step consists in ensuring that what the expert wants to express can be actually expressed. The expert usually explains the main concepts she wants to handle to the modeler. Testing a metamodel can be done in a iterative process [13]. As soon as a first draft has been erected, it can be used to generate models automatically. These generated models are shown to the expert. She then decides if they are or not part of the correct modeling space (oracle). During the next steps, the annotated models are used to refine the structure of the metamodel. Models are generated after each refinement exhibiting the corner cases of the metamodeling space and the expert annotates them again. This iteration is repeated until no more invalid models are generated, *i.e.,* the metamodel fits to the application space.

In this section, we show how our generic framework can be applied to metamodel testing and evaluate this application. With respect to the objectives, we consider the whole metamodel to evaluate the coverage, *i.e.,* we tag all the elements as mandatory. Indeed, as the metamodel defines the modeling space, we have to select a representative set of models that covers this space. For the minimality, we study each objective separately and the two-by-two combinations.

4.1 Evaluation

Our approach is implemented in Java using Eclipse Modeling Framework (EMF) to ease the use of complex metamodels written in Ecore. Model instances are encoded in XMI (XML METADATA INTERCHANGE).

In the remainder of this section, we present our research questions and the experimental setup.

4.1.1 Research questions

RQ1: *Are the results of our approach attributable to the search strategy or to the number of explored solutions?* We answer this question by exploring the same amount of solutions by our algorithm and by a random search, and compare the best solutions from both strategies.

RQ2: *Is our approach better than a mono-objective multi-criteria search?* To answer this question, we compare the results of our approach to those of a classical genetic algorithm with a single objective that combines the coverage and minimality criteria.

RQ3: *What combination of the minimality criteria gives the best tradeoff between coverage and solution size?* We answer this question by running both our approach and the mono-objective algorithm with all combinations of one or two minimality criteria, and compare the resulting alternatives. We did not consider the three minimality criteria at the same time since, with NSGA-II, it is difficult to converge towards interesting solutions with four objectives (including the coverage).

4.1.2 Experimental setup

Metamodels To assess our selection process, we executed the algorithms on three different metamodels: two small ones, Fea-

ture Diagram which contains 5 classes and 8 features, and Composite State-Machine which contains 4 classes and 10 features; and a larger one, ATL2.0, with 84 classes and 146 features.

Model base. As we are using a fixed model base to select the representative model sets, we have to ensure the diversity of the models in that base. Therefore, we run the Instantiator to produce 10.000 models with different structural characteristics. During the generation, we varied the expected size of models from 2 to 200 classes (with steps of 10), and we picked randomly the numbers of attributes and references in a range of 0 to 14. The depth of reference chains is picked between 0 and 10. We took those numbers from common knowledge about the statistical structure of metamodels and its correlation with the practical use. These parameters can be changed in our framework. At the end, the 10.000 models generated are stored in the base from which the initial population of solutions is created and from which the models are randomly picked for the mutation operator. During the execution, even if our approach allows to specify which range of size the models must have to be picked, we took all the models present in the base.

Best solution selection. A multi-objective algorithm gives a set of near-optimal solutions (Pareto set). In an application scenario, an expert decides which solution to select. However, for our evaluation, we have to choose a solution among the Pareto set to compare our algorithm with those producing a single solution. Choosing such a solution in a multi-objective setting is a well-known problem as explained by Murashkin *et al.* [26]. In these experiments, we select the solution having the lowest Euclidian distance with the ideal solution having all objectives equal to 1. Additionally, as all the studied algorithms are probabilistic by nature, we executed each algorithm 30 times and compare the distributions of the results.

Algorithmic parameters. When parent solutions are selected, the crossover and mutation operators are applied with a certain probability. High mutation probability on a solution with a few models would have a dramatic impact, *i.e.,* changing one model in a set of two models would results in a big variation of the coverage. However, such a probability would have a limited impact for solutions with a lot of models. As generally the average size of solutions is correlated with the size of the metamodel, we considered different mutation probabilities: 0.7 for the largest metamodel ATL2.0, and 0.35 for the smallest metamodels Feature Diagram and Composed State-Machine. The crossover probability is set to 0.9 in all cases. We ran both the mono- and multi-objective algorithms with a population size of 100 model sets during 800 generations.

5. EVALUATION RESULTS

For the small metamodels, all the approaches produced solutions with high coverage and a very few models. For questions RQ1, RQ2, and RQ3, we did not observe any notable difference. The results reported in the following paragraphs are only for the largest metamodel ATL2.0.

5.1 RQ1: comparison between our approach and a random search algorithm

Results of the comparison between a random selection and our approach are shown in Figures 8 and 9. Random selection builds sets by randomly picking models from the model base. To have a fair comparison, the random selection produces 100×800 model sets, which corresponds to the number of model sets explored by our approach (800 generations of 100 model sets each). The size of the set is randomly set (between 0 and 40) for each iteration of the random selection and for the initial population in our approach. As it can be seen in Fig. 8, column 1, the coverage for 30 random executions is by far lower (an average of 66%) than the one of our approach (an average of 91% for the best case and 83% for the worst case). Moreover, our algorithm produces uniform results (flat boxplot) compared to the random search, for which the coverage varies from 44% to 76%. Both the random search and our approach have the best solutions with small model sets (around 18 models per solution) as shown in Fig. 9, column 1.

5.2 RQ2: comparison between our approach and mono-objective

Now that we established that the quality of results is attributable to our search strategy and not to the number of explored solutions, the next step if to assess if a multi-objective search brings a benefit compared to a mono-objective search. For the mono-objective approach, we use a classical genetic algorithm with a single-objective function, defined as the average of the coverage and the minimality values. Formally:

$$f_{mono}(s) = \frac{COV(s) + minimality(s)}{2} \quad (5)$$

where $minimality(s)$ can be DIS, or MIN functions as stated in, respectively, equations 1 and 3. Additionally, we use the crossover and mutation operators of our multi-objective approach with the same probabilities.

In Fig. 8 and Fig. 9, columns two and three present the result of a mono-objective algorithm using a combination of the coverage with, respectively, DIS and MIN. Columns four and five give the results for respectively the same configurations of our approach.

The mono-objective algorithm produces a higher coverage value when DIS is used (93% on average compared to 88%). However, this comes at the cost of having very large model sets (70 on average compared to 13 for the multi-objective counterpart). When MIN is used our approach produces a higher coverage (91% on average compared to 87%) with almost the same number of models and the same number of objects per model.

In conclusion, when considering the coverage with a minimality criterion, our approach achieves the best tradeoff between the coverage and the minimality for model set selection. The coverage can be higher with the mono-objective algorithm but with very large solutions.

5.3 RQ3: comparison between combinations of the minimality objectives

Different combinations of minimality objectives result in different trade-offs between the coverage and the size of the solutions. As shown in Fig. 8 and Fig. 9, when selecting a unique criterion (columns four to six), MIN and MIN-R have better coverage values than DIS with almost the same number of models in the solutions. However, DIS tends to have smaller models as shown by the boxplots of the average number of objects. When considering two minimality criteria (columns seven to nine), the combination of DIS with MIN gives the best results in terms of coverage but also in terms of the model average size. The only weak point is a slightly higher number of models in the solutions.

In conclusion, there is not a clear winner for the minimality criteria combinations. The combination of DIS and MIN seems, however, more promising than the others.

5.4 Performance

As stated in Section 5.1, we ran the experiment 30 times for each combination for 800 generations having, each, 100 solutions. The computer used is a classic desktop with an Intel(R) Core(TM) i7-4770 @ 3.40 GHz with 32 Go RAM. The execution of our algorithm takes less than a minute with small metamodels. With the

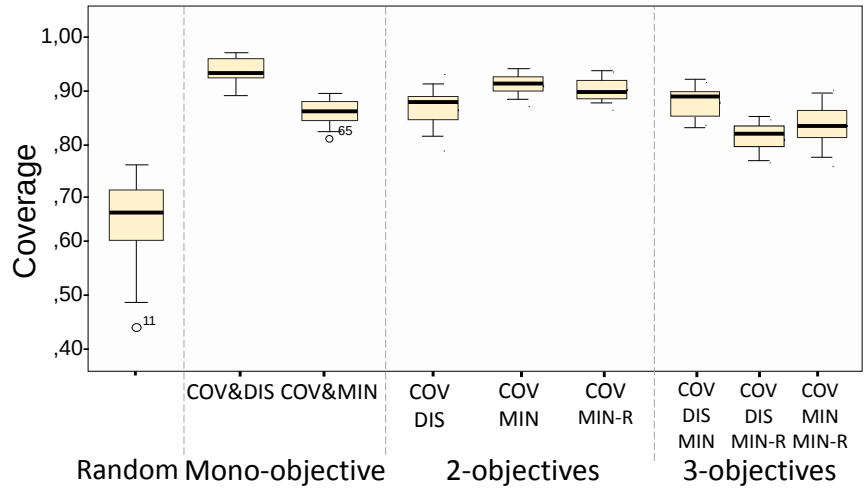

Figure 8: Coverage for ATL2.0

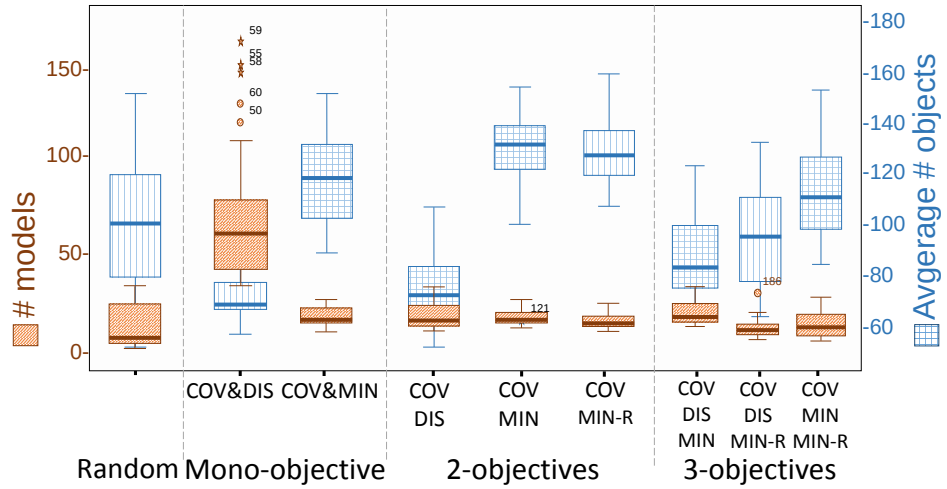

Figure 9: Size of solutions for ATL2.0

larger metamodel (*i.e.,* ATL2.0), the execution time depends on the algorithm and the objectives chosen, as shown in Fig. 10.

When considering one minimality criterion and with the mono-objective strategy (columns two and three), DIS makes the execution time more volatile, although the median is almost the same for DIS and MIN (around 15 minutes). This can be explained by the fact that DIS tends to select solutions with many models that have to be compared pairwise as explained in Section 3.1.2.

With the multi-objective strategy, (columns four to six), the execution proceed faster, especially when MIN or MIN-R are considered (around 10 minutes). The execution time is almost doubled when considering two minimality criteria (around 21 minutes) as it can be seen in columns seven to nine. In addition to the cost of evaluating an additional objective, an extra-cost is brought by the dominance computation to define the fronts.

These figures are acceptable since the process of generating instances does not require a real-time execution. Still to reduce this time, we experimented with 500 generations (instead of 800). With this setting, we almost cut the execution time by half while limiting the coverage degradation to a very small amount. In this context, we improved our framework by allowing the user to achieve a good compromise between the execution time and the coverage by offering her the possibility of monitoring the coverage evolution and of stopping the execution when results are judged acceptable.

5.5 Discussion

The above-mentioned observations are statistically significant, *i.e.,* Mann-Whitney test with a *p-value* < 0.05. The significance and effect size of the tests used to answer the research questions are summarized in table 1. Results show that our approach is a better alternative than random and mono-objective strategies for model case selection for the metamodeling activity. However, there is no clear best combination of the minimality objectives.

Our framework is generic enough to consider different definitions of the coverage (intentional definition) together with a selected composition of minimality definitions (extensional definition). We have, however, to acknowledge the following limitations. Firstly, we generate the model base without taking into consideration the characteristics of the studied metamodel. Small metamodels may require model bases with relatively small models, whereas large metamodels may need large bases. It will be interesting in the future to ensure the diversity/variability of our model base with re-

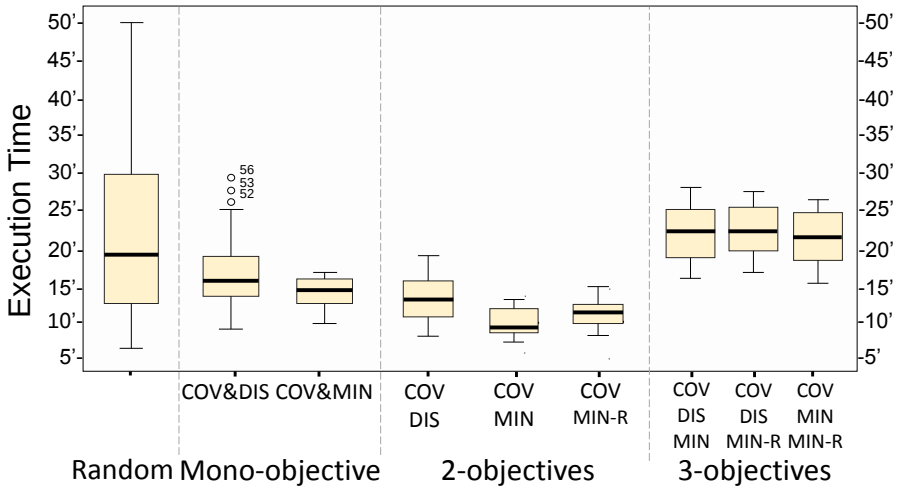

Figure 10: Time elapsed during executions

	RQ1 col. 1 vs col. 4		RQ2 col. 2 vs col. 4		RQ3 col. 4 vs col. 7	
	p-value	*Cohen's d*	*p-value*	*Cohen's d*	*p-value*	*Cohen's d*
Coverage	< 0.01	High	< 0.01	High	0.86	Low
Size in number of models	< 0.01	High	< 0.01	High	0.83	Low
Size in average number of objects	< 0.01	High	0.530	Low	< 0.01	high
Execution time	< 0.01	High	< 0.05	Medium	< 0.01	Low

Table 1: Statistical significance and effect size for the differences between alternatives of col. 1 (random), col. 2 (mono-objective with DIS), col. 4 (multi-objective with DIS), and col. 7 (multi-objective with DIS and MIN)

spect to the considered metamodels and tasks, and to consider the situation where the model base is gathered from existing models. Moreover, we plan to consider more large metamodels to study the generalizability of our approach.

Another important aspect is to evaluate the impact of the selected model sets on the quality of the targeted task. For example, when testing a transformation, one can measure the number of errors found thanks to the selected set of model cases. Similarly, when learning WFRs, we can evaluate the correctness of the WFRs obtained by a set of selected model cases.

Finally, we do not handle the case of manual completion after selecting a set of models. We believe that we can define common aspects of this activity to support the expert work.

6. RELATED WORK

Our contribution crosscuts different research fields: (1) model/instance generation/selection, (2) testing MDE tasks, including search-based testing, and (3) learning MDE artifacts by examples. In the remainder of this section, we discuss some of the existing work in those fields.

For model instance generation, Wu discussed, in a literature review [25], algorithms used and fields investigated for/in model instance generation. His conclusion, which reflects a general trend in the modeling literature, states that instance generation is mostly investigated in order to feed model transformation testing algorithms. He does not mention multi-objective model generation, nor metamodel testing as a goal for the generation process. Most of the existing work on instance generation, *e.g.,* [27, 28], target specific tasks and purposes without a generalization effort. In particular, Ehrig *et al.* [29] propose an approach to translate metamodels into *graph-grammars* in order to use them to generate model instances. Although, the authors give the principles of the model generation, they did not implement this part nor validate it in terms of coverage. Other limits of this approach, i.e., completeness and rule complexity, are pointed out by Hoffmann *et al.* [30]. Other teams use constraint logic programming to derive instances [31, 32]. Metamodels are translated into constraint satisfaction problems and SAT/SMT solvers are used to find an instance conform to the metamodel, or to prove that no instance can be found. Here again, the coverage is not targeted by these approaches. Gonzales-Perez *et al.* [33] use a similar approach for the verification of EMF models. Finally, Sen *et al.* [12] use *Alloy* [34] to generate instances satisfying a translated version of the metamodel. However, the UML to Alloy translation is restrictive and challenging as pointed-out in [35].

As these studies do not address the same problem, it is difficult to compare our approach with them. Indeed, we propose a generic model set generation framework that can be applied to various MDE artifacts and tasks, with the concern of minimizing the size of the generated sets.

In MDE testing, an extensive body of research work has focused on model transformation. On the other hand, metamodel testing has gathered less attention. Selim *et al.* [36], organize the transformation testing process into four phases: producing the test cases, assessing the test cases, producing the oracle function, and performing the test. To evaluate a test suite, authors use mainly a coverage criterion applied on the transformation's input space (see, for instance, [9, 10, 27]). Gogolla *et al.* [28] decompose the coverage using classifying terms. Mutation analysis is another way to assess the generated input cases as described [37, 38]. Guerra *et al.* [39] take the specification as a guide for test generation. Finally, oracle

definition is an open topic, and many oracle functions have been proposed (see, for example, the recent work by Finot *et al.* [40]). In these research contributions, a clear focus is put on the coverage definition. However, the different possibilities to achieve the minimality of test suites are rarely discussed.

The other family of work in MDE testing target the metamodels. Obviously, the accuracy of a metamodel must be assessed in order to reason about its modeling space. This vision is supported by Sadilek *et al.* [8] who consider that the field is not investigated enough. To test metamodels, Cadavid *et al.* [4] explore the boundaries of the modelling space using an mono-objective evolutionary algorithm. Other teams focus on the interaction between modellers and stakeholders [13]. Both exhibit the need to have representative, yet small, sets of testing models.

The work of Cadavid *et al.* [4] falls in the category of search-based testing. Search-based testing aims at defining one or more functions that capture the testing objectives [41]. Search-based test case generation is one of the most active fields in the search-based software engineering community for many resulting in many approaches and tools. A quick look at the SBSE repository [42] shows that more than 700 papers in SBSE, published between 1975 and 2015, are dealing with testing and/or debugging, and a large portion of these papers relates to test case generation. Nonetheless, as pointed out by Harman *et al.*in a recent study [43], multi-objective search-based approaches are still scarce when it is clear that search objectives in testing are contradictory (*i.e.,* coverage and minimality) and cannot be combined efficiently into a unique fitness function.

The final family of research we discuss in this section is concerned with the learning of MDE artifacts from examples. Learning by examples is a relatively new field [3] which finds its root in the idea of a generative perspective on programming [44]. Model transformation is the most studied field with the early work from Balogh *et al.* [45] and Wimmer *et al.* [2]. In these papers, the authors aim at abstracting mappings between two metamodel starting from examples of source and target models. These mappings can then be transformed into transformation programs as done by Saada *et al.* [46]. Similarly, Sun *et al.* [47] investigated how to learn transformations by means of *demonstrations* made by experts. Demonstration actions are modeled as examples from which authors infer the transformation knowledge. Kessentini *et al.* [48] learn model transformations from example by analogy. They do not try to abstract the transformation knowledge, but rather propose a concrete transformation for a given source model. More recently, Faunes *et al.* [6] and Baki *et al.* [19, 18], learn directly the code of transformations from the example. In addition to learn transformations, examples were used to learn well-formedness rules by Faunes *et al.* [5]. Finally, examples are used for manual modeling and metamodeling activities [49, 14].

7. CONCLUSION

In this paper, we propose a generic framework to automatically select model sets for various MDE tasks and for different purposes. We model the model set selection as a multi-objective optimization problem, and we solve it using the evolutionary algorithm NSGA-II. Our framework can be customized for a specific task and purpose by giving indications about the coverage criterion, which will be later automatically assessed. The second customization aspect consists in choosing one or more pre-defined minimality criteria.

We illustrate the use of our framework on the metamodeling task with the testing purpose. We evaluated this application on small and large metamodels and showed that the multi-objective strategy offers a better alternative to random or mono-objective search.

As a future work, we are studying the impact of the selected model sets on the efficiency of testing or learning MDE tasks.

8. REFERENCES

[1] D. C. Schmidt, "Model-driven engineering," *IEEE Computer Society*, vol. 39, no. 2, 2006.

[2] M. Wimmer, M. Strommer, H. Kargl, and G. Kramler, "Towards model transformation generation by-example," in *40th Hawaii Int. Conf. on Systems Science*, 2007, p. 285.

[3] K. Bąk, D. Zayan, K. Czarnecki, M. Antkiewicz, Z. Diskin, A. Wąsowski, and D. Rayside, "Example-driven modeling: Model = abstractions + examples," in *Proc. of the Int. Conf. on Software Engineering*, 2013, pp. 1273–1276.

[4] J. J. Cadavid, B. Baudry, and H. A. Sahraoui, "Searching the boundaries of a modeling space to test metamodels," in *Proc. of the Int. Conf. on Software Testing Verification and Validation*, 2012, pp. 131–140.

[5] M. Faunes, J. Cadavid, B. Baudry, H. Sahraoui, and B. Combemale, "Automatically searching for metamodel well-formedness rules in examples and counter-examples," in *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, 2013, pp. 187–202.

[6] M. Faunes, H. Sahraoui, and M. Boukadoum, "Genetic-programming approach to learn model transformation rules from examples," in *Proc. of the Int. Conf. on Theory and Practice of Model Transformation*, 2013, vol. 7909, pp. 17–32.

[7] M. Kessentini, W. Kessentini, H. Sahraoui, M. Boukadoum, and A. Ouni, "Design defects detection and correction by example," in *Proc. of the Int. Conf. on Program Comprehension*, 2011, pp. 81–90.

[8] D. Sadilek and S. Weissleder, "Testing metamodels," in *Proc. of the Eur. Conf. on Modelling Foundations and Applications*, 2008, vol. 5095, pp. 294–309.

[9] E. Brottier, F. Fleurey, J. Steel, B. Baudry, and Y. le Traon, "Metamodel-based test generation for model transformations: an algorithm and a tool," in *Int. Symp. on Software Reliability Engineering, 2006. ISSRE '06. 17th*, 2006, pp. 85–94.

[10] F. Fleurey, B. Baudry, P.-A. Muller, and Y. Le Traon, "Towards dependable model transformations: Qualifying input test data," *Int. J. on Soft. and Systems Modeling*, vol. 8, no. 2, pp. 185–203, 2009.

[11] J. Mottu, S. Sen, M. Tisi, and J. Cabot, "Static analysis of model transformations for effective test generation," in *Int. Symp. on Software Reliability Engineering*, 2012, pp. 291–300.

[12] S. Sen, B. Baudry, and J. Mottu, "Automatic model generation strategies for model transformation testing," in *Proc. of the Int. Conf. on Theory and Practice of Model Transformation*, 2009, vol. 5563, pp. 148–164.

[13] J. Sanchez-Cuadrado, J. de Lara, and E. Guerra, "Bottom-up meta-modelling: An interactive approach," in *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, 2012, vol. 7590, pp. 3–19.

[14] J. J. Lopez-Fernandez, J. S. Cuadrado, E. Guerra, and J. de Lara, "Example-driven meta-model development," *Int. J. on Soft. and Systems Modeling*, pp. 1–25, 2013.

[15] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, 2012.

[16] B. Baudry, S. Ghosh, F. Fleurey, R. France, Y. Le Traon, and J.-M. Mottu, "Barriers to systematic model transformation testing," *Communications of the ACM*, vol. 53, no. 6, pp. 139–143, 2010.

[17] M. Kessentini, H. Sahraoui, and M. Boukadoum, "Example-based model-transformation testing," *Automated Soft. Eng.*, vol. 18, no. 2, pp. 199–224, 2011.

[18] I. Baki and H. Sahraoui, "Multi-step learning and adaptive search for learning complex model transformations from examples," *ACM Trans. on Soft. Eng. and Methodology*, vol. X, p. 36, 2015.

[19] I. Baki, H. Sahraoui, Q. Cobbaert, P. Masson, and M. Faunes, "Learning implicit and explicit control in model transformations by example," in *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, 2014, vol. 8767, pp. 636–652.

[20] C. Jeanneret, M. Glinz, and B. Baudry, "Estimating footprints of model operations," in *Proc. of the Int. Conf. on Software Engineering*, May 2011, pp. 601–610.

[21] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Proc. of the Int. Conf. on System Testing Verification and Reliability*, vol. 22, no. 5, pp. 297–312, August 2012.

[22] J. J. Cadavid Gómez, "Assistance à la méta-modélisation précise," Ph.D. dissertation, Rennes 1, France, 2012, thèse de doctorat dirigée par Jézéquel, Jean-Marc et Baudry, Benoit.

[23] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II," in *Int. Conf. on Parallel Problem Solving from Nature - PPSN*, 2000.

[24] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," *Commun. ACM*, vol. 31, no. 6, pp. 676–686, 1988.

[25] H. Wu, R. Monahan, and J. F. Power, "Metamodel instance generation: A systematic literature review," *CoRR*, vol. abs/1211.6322, 2012.

[26] A. Murashkin, M. Antkiewicz, D. Rayside, and K. Czarnecki, "Visualization and exploration of optimal variants in product line engineering," in *Int. Software Product Line Conference*. ACM, 2013, pp. 111–115.

[27] C. A. Gonzalez and J. Cabot, "Test data generation for model transformations combining partition and constraint analysis," in *Proc. of the Int. Conf. on Theory and Practice of Model Transformation*, 2014, vol. 8568, pp. 25–41.

[28] M. Gogolla, A. Vallecillo, L. Burgueno, and F. Hilken, "Employing classifying terms for testing model transformations," in *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, 015, pp. 312–321.

[29] K. Ehrig, J. Koster, G. Taentzer, and J. Winkelmann, "Generating instance models from meta models," in *Formal Methods for Open Object-Based Distributed Systems*, 2006, pp. 156–170.

[30] B. Hoffmann and M. Minas, "Defining models - meta models versus graph grammars," *ECEASST*, vol. 29, 2010.

[31] A. Ferdjoukh, A.-E. Baert, A. Chateau, R. Coletta, and C. Nebut, "A csp approach for metamodel instantiation," in *Int. Conf. on Tools with Artificial Intelligence*, 2013.

[32] H. Wu, "Generating metamodel instances satisfying coverage criteria via smt solving," in *Proc. of the Int. Conf. on Model-Driven Eng. and Soft. Development*, 2016, pp. 40–51.

[33] C. A. González Pérez, F. Buettner, R. Clarisó, and J. Cabot, "EMFtoCSP: A Tool for the Lightweight Verification of EMF Models," in *Formal Methods in Soft. Eng.: Rigorous and Agile Approaches (FormSERA)*, 2012.

[34] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[35] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, *UML2Alloy: A Challenging Model Transformation*. Springer Berlin Heidelberg, 2007, pp. 436–450.

[36] G. M. K. Selim, J. R. Cordy, and J. Dingel, "Model transformation testing: The state of the art," in *Workshop on the Analysis of Model Transformations*, 2012, pp. 21–26.

[37] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. on Soft. Eng.*, vol. 37, no. 5, pp. 649–678, 2011.

[38] V. Aranega, J.-M. Mottu, A. Etien, T. Degueule, B. Baudry, and J.-L. Dekeyser, "Towards an automation of the mutation analysis dedicated to model transformation," *Software Testing, Verification and Reliability*, vol. 25, no. 5-7, pp. 653–683, 2015.

[39] E. Guerra, "Specification-driven test generation for model transformations," in *Proc. of the Int. Conf. on Theory and Practice of Model Transformation*, 2012, pp. 40–55.

[40] O. Finot, J.-M. Mottu, G. Sunyé, and C. Attiogbé, "Partial test oracle in model transformation testing," in *Proc. of the Int. Conf. on Theory and Practice of Model Transformation*, 2013, pp. 189–204.

[41] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. Mcminn, "An orchestrated survey of methodologies for automated software test case generation," *J. Syst. Softw.*, vol. 86, no. 8, pp. 1978–2001, 2013.

[42] Y. Zhang, M. Harman, and A. Mansouri, "The SBSE repository: A repository and analysis of authors and research articles on search based software engineering."

[43] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *Proc. of the Int. Conf. on Software Testing Verification and Validation*, 2015, pp. 1–12.

[44] K. Czarnecki, E. Ulrich, and P. Steyaert, "Beyond objects: Generative programming," in *ECOOP'97 Workshop on Aspect-Oriented Programming*, Jyväskylä, Finland, 1997.

[45] Z. Balogh and D. Varró, "Model transformation by example using inductive logic programming," *Int. J. on Soft. and Systems Modeling*, vol. 8, no. 3, pp. 347–364, 2009.

[46] H. Saada, X. Dolques, M. Huchard, C. Nebut, and H. A. Sahraoui, "Generation of operational transformation rules from examples of model transformations," in *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, 2012, pp. 546–561.

[47] Y. Sun, J. White, and J. Gray, "Model transformation by demonstration," in *Proc. of the Int. Conf. on Model-Driven Engineering Languages and Systems*, 2009, pp. 712–726.

[48] M. Kessentini, H. Sahraoui, M. Boukadoum, and O. B. Omar, "Search-based model transformation by example," *Int. J. on Soft. and Systems Modeling*, vol. 11, no. 2, pp. 209–226, 2010.

[49] D. Zayan, M. Antkiewicz, and K. Czarnecki, "Effects of using examples on structural model comprehension: a controlled experiment," in *Proc. of the Int. Conf. on Software Engineering*, 2014, pp. 955–966.