Université de Montréal

From Examples to Knowledge in Model-Driven Engineering: a Holistic and Pragmatic Approach

par

Edouard Batot

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique - Génie Logiciel

novembre 2018

Sommaire

Le Model-Driven Engineering (MDE) est une approche de développement logiciel qui propose d'élever le niveau d'abstraction des langages afin de déplacer l'effort de conception et de compréhension depuis le point de vue des programmeurs vers celui des décideurs du logiciel. Cependant, la manipulation de ces représentations abstraites, ou modèles*, est devenue tellement complexe que les moyens traditionnels ne suffisent plus à automatiser les différentes tâches.*

De son côté, le Search-Based Software Engineering (SBSE) propose de reformuler l'automatisation des tâches du MDE comme des problèmes d'optimisation. Une fois reformulé, la résolution du problème sera effectuée par des algorithmes métaheuristiques. Face à la pléthore d'études sur le sujet, le pouvoir d'automatisation du SBSE n'est plus à démontrer.

C'est en s'appuyant sur ce constat que la communauté du Example-Based MDE (EB-MDE) a commencé à utiliser des exemples d'application *pour alimenter la reformulation SBSE du problème d'apprentissage de tâche MDE. Dans ce contexte, la concordance de la sortie des solutions avec les exemples devient un baromètre efficace pour évaluer l'aptitude d'une solution à résoudre une tâche. Cette mesure a prouvé être un objectif sémantique de choix pour guider la recherche métaheuristique de solutions.*

Cependant, s'il est communément admis que la représentativité des exemples a un impact sur la généralisabilité des solutions, l'étude de cet impact souffre d'un manque de considération flagrant. Dans cette thèse, nous proposons une formulation globale du processus d'apprentissage dans un contexte MDE incluant une méthodologie complète pour caractériser et évaluer la relation qui existe entre la généralisabilité des solutions et deux propriétés importantes des exemples, leur taille et leur couverture.

Nous effectuons l'analyse empirique de ces deux propriétés et nous proposons un plan détaillé pour une analyse plus approfondie du concept de représentativité, ou d'autres représentativités.

Mots clefs : Génie Logiciel · Génie Logiciel Expérimental · Ingénierie Dirigée par les Modèles · Apprentissage Machine · Apprentissage par les Exemples

Summary

Model-Driven Engineering (MDE) is a software development approach that proposes to raise the level of abstraction of languages in order to shift the design and understanding effort from a programmer point of view to the one of decision makers. However, the manipulation of these abstract representations, or models*, has become so complex that traditional techniques are not enough to automate its inherent tasks.*

For its part, the Search-Based Software Engineering (SBSE) proposes to reformulate the automation of MDE tasks as optimization problems. Once reformulated, the problem will be solved by metaheuristic algorithms. With a plethora of studies on the subject, the power of automation of SBSE has been well established.

Based on this observation, the Example-Based MDE community (EB-MDE) started using application examples *to feed the reformulation into SBSE of the MDE task learning problem. In this context, the concordance of the output of the solutions with the examples becomes an effective barometer for evaluating the ability of a solution to solve a task. This measure has proved to be a semantic goal of choice to guide the metaheuristic search for solutions.*

However, while it is commonly accepted that the representativeness of the examples has an impact on the generalizability of the solutions, the study of this impact suffers from a flagrant lack of consideration. In this thesis, we propose a thorough formulation of the learning process in an MDE context including a complete methodology to characterize and evaluate the relation that exists between two important properties of the examples, their size and coverage, and the generalizability of the solutions.

We perform an empirical analysis, and propose a detailed plan for further investigation of the concept of representativeness, or of other representativities*.*

Keywords: Software Engineering · Expérimental Software Engineering · Model Driven Engineering · Machine Learning· Learning from Examples

Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to thank Prof. Houari Sahraoui, my supervisor and colleague. The narrations plural conjugation of this thesis is here to greet his incredible benevolence, his tireless diplomacy, and his flawless bonhomie. Your attention figures a milestone on the path I follow.

Then, I would like to thank in the language they read my parents, Maurice et Sylviane, deux êtres pour qui il ny aura jamais assez de place pour exprimer ma gratitude envers leur appui et leur confiance inébranlables dans toutes mes entreprises.

Humongous thanks to Vasco Sousa for the memorable time spent brainstorming on the whiteboard — to think of insolvent problems is to calibrate our understanding, as he coins it. I would like to mention as well Chihab Eddine Mokaddem and Wael Kessentini, close colleagues, always supporting and motivating.

To name but a few, I would like to mention Yasmin, Yacine, and Youssef for the inexhaustible source of imagination they feed profusely. As well, thanks to professors Doizi, Fiema, and Lepage for their warm and insightful input on pedagogy and linguistics. Pauline, Noël, and Mathieu, *bien sûr*, dear friends from out there, always here. And Zahra, for sharing her sharp understanding of the broader picture that frames it all.

And to all whose names remain in the shade, comrades and friends, family and neighbors, colleagues and collaborators, whose indefectible support made this investigation possible, I dedicate *un grand merci*.

To the reader finally. Could you find some references for your work, some food for your thoughts, or some combustible to feed your bonfire — I will be glad to discuss the matter and look forward to hearing from you.

My two fears are distortion and accuracy,
or rather the kind of accuracy produced
by too dogmatic a generality
and too posistivistic a localized focus.

EDWARD W. SAID, IN
'*ORIENTALISM'* (1978)

Chapter 1

in the presence of extraordinary actuality,
consciousness takes the place of imagination

WALLAS STEVENS, IN
'*OPUS POSTHUMOUS*' (1957)

Introduction

All engineering fields employ abstract representations, or *models*. They are useful for understanding and simulating complex structures and processes when their execution is not wanted or possible. Yet, the Software Engineering (SE) community has always considered models as a secondary artefact. Utility of the models is recognized for the initial understanding of a problem and for drafts of the system under development. They show interesting new points of view and figure what the system *could be* but they are quickly left out and fall into disuse with the advancement of the development process. In other cases, their discrepancy takes over, avoiding modelling artifacts to cope with the evolution of the software.

Notwithstanding this limitation, the SE community has been striving to address a rising complexity and a growing need for productivity by means of abstraction. From machine code, via structural languages and functional languages to object-oriented languages, and domain-specific languages, the task of writing software has shifted from a programmer perspective to the broader one of stakeholders (Whittle et al, 2014). Yet, at first glance, the literature on the matter shows that SE offers a medium of choice to support and facilitate not only the representation of the problem under study, but also the representation of the software that will solve it (Bézivin, 2005). In addition, the effort of automation required to link the two is now offered by an incredible computational power. All the ingredients are there (Mussbacher et al, 2014).

1. (Meta)Modelling

Unstructured models, such as sketches and hand-written graphic representations are useful to enhance communication between stakeholders (Harman et al, 2012). Yet, as Williams in his dissertation kindly reminds the reader, with the advent of Model-Driven Engineering (MDE), models are not considered anymore like mere documentation artifacts but rather as an effective force in the software development process (Williams, 2013). In order to bring tangible value to automatic processes involving abstract representations, their translation into effective executable code must be automated. Indeed, when models are well-defined (*i.e.,* a *meta*definition is provided), automated processes can interpret their structure and include them. The conversion from MDE abstract representations to concrete implementation is done using a sequence of automated transformations that generate, from a high-level model, the executable low-level code (Sendall and Kozaczynski, 2003; Umuhoza et al, 2015). Thereby, MDE enables a focus on models and moves the development effort from developers to experts who can in return express themselves directly in the esoteric language dedicated to their domain. This shift leads to a gain in productivity and a reduction of time-to-market (Selic, 2003, 2012). In practice, using MDE arguably speeds up responses to requirement changes, facilitates communication with stakeholders, and increases portability, maintainability, and of course productivity (Hutchinson et al, 2011a,b).

MDE adoption faces a limitation. As mentioned above, the construction of Domain Specific Languages (DSL) can benefit SE only if the tasks supporting the manipulation of models are automated. The problem of automation in MDE is twofold. On the one hand, writing MDE artifacts manually is arduous since it requires knowledge of different natures: both a strong skillset in modelling languages, which is essential, and an expertise specific to the application domain. On the other hand, the very specific nature of problems that DSLs focus on avoids amassing the information required for automation in the traditional fashion (Babur et al, 2018).

2. Metaheuristics

Metaheuristics builds on the assumption that it is easier to check that a candidate solution solves a problem than to actually construct a solution to that problem (Luke, 2013). To alleviate the limitation that MDE automation faces, Search Based Software Engineering

(SBSE) research makes use of metaheuristic techniques to solve complex SE problems (Harman and Jones, 2001). It therefore stands that the reformulation of a software engineering (SE) problem into a Search Optimization (SO) problem is time and effort worthy (Harman et al, 2012). It also provides a solid support to automate the learning of complex MDE tasks.

From SE to SO, the reformulation consists of only two constituents: a representation of the problem solution space and a fitness function (Harman et al, 2012, 2015). The representation describes the problem at hand, its specificities and constraints. A fitness function evaluates candidate solutions. In measurement theory lexica, a fitness function merely imposes an ordinal scale of measurement upon the individual solutions sufficient to know which of two solutions surpasses the other. SBSE uses metaheuristics to find (near) optimal solution(s), *i.e.,* solutions that fall within a specified acceptable tolerance.

The growing interest in the field, with more than 1700 authors participating in the SBSE repository (Zhang et al, 2014), as well as the number of areas to which SBSE techniques have been applied, show how conceptually simple is the reformulation of SE problem into SBSE problem (Boussaïd et al, 2017). For our matter, MDE benefits as well from this statement. We will see how SBSE techniques are useful in addressing the complexity involved in the automation of MDE tasks.

3. Learning from examples

To our matter, MDE makes it possible to illustrate the expected behavior of a task through examples of applications. Indeed, an example that is made of a potential input for that task (a model), coupled with its corresponding output once executed (a model as well), captures some of the semantics in a black box fashion. The conformance of a solution to a set of application examples can be used to measure the semantic relevance of that solution. Therefore, to learn a MDE task, semantics are captured in an example set and the relevance of a solution is employed as an objective to guide the exploration of the search space. These semantics are "learned" using SBSE techniques.

On the one hand, the result is an incredible automation potential. Experts do not need to write the task in MDE language, they can simply provide examples that show how is that task expected to operate. On the other hand, it also exhibits a strong dependence on the quality of the examples that were elected to train the algorithm. To convince a broader

audience, automation needs a safeguard. We believe that a lack of precise definition of that quality hinders the adoption of MDE and we propose to address this issue in the present thesis.

4. Motivation

More precisely, we see a bottleneck in the process of learning at the very production of application examples. Indeed, how can we assess that a set of examples is representative of a problem? And if representativeness cannot be defined exactly, how can we measure the generalization power of solutions derived with such learning technologies? In other words, how can we assess that the selected sample (of application examples) illustrates all possible inputs a task might need to process? It seems to us essential to rationalize the representativeness of examples in order to avoid the obvious risk of overfit that their limited number implies. We see this way of estimating the power of the generalization of solutions produced by learning algorithms as a means to push back the limitations faced by EB-MDE research.

However, it is strange that, to our knowledge, research teams investigating Example Based learning have been addressing this requirement by using examples precisely *apt* to show the benefits of their study. They are either built *ad hoc*, or picked from the industry. The former method conveys concise and comprehensible information; the latter gives a proof for scalability and some kind of reliability. In all cases, the examples representativeness is not systematically evaluated (Batot, 2015).

5. Thesis contributions

In this thesis, we propose a methodology to account for a meaningful characterization of application example sets to foster Example-Based automation in Model Driven Engineering (EB-MDE). We believe that there exists a relation between the representativeness (estimated by the coverage) of examples used to train an algorithm, and the quality (estimated by the relevance of the training set) of the resulting solutions. We propose a methodology and tools to control the production of examples of quality and to derive the knowledge they embody. We illustrate our words with a direct improvement of the state-of-the-art technic

for the automatic learning of wellformedness rules (WFR), and we assess our approach with a meticulous empirical study.

Here, is a list of our contributions:

(1) **A formalization of the thorough learning process** in three key steps: the generation of partial examples, their manual completion, and the execution of the learning by examples algorithm;

(2) **A generic framework for model-set selection for learning or testing Model-Driven Engineering tasks** – tailored for example sets generation, with configurable coverage criteria and minimality;

(3) **A significant improvement of one of the most widely spread algorithm for multi-objective optimization, NSGA-II**, with the injection of a social dimension to the measurement of diversity among a population of solutions;

(4) **A pragmatic characterization of examples** – assessed with an empirical study of the relation between example sets coverage and size, and the accuracy of the automatic learning of WFR.

6. Thesis structure

The thesis is structured as follows. In the first chapter, we depict the general background of our investigation and introduce MDE and SBSE more thoroughly. We then depict related studies on learning from examples automation as well as model generation. In Chapter 3, we portray a big picture of our investigation, the general research map and its landmarks. Chapter 4 presents our first contribution. We detail the selection of model sets and a case study illustrating its benefits. The following article (5) presents our second contribution. We modified the classic multi-objective non-sorting genetic algorithm NSGA-II (Deb et al, 2000) by injecting a social dimension in the evaluation of populations diversity. We will detail the idea, an implementation and a complete empirical evaluation. Finally, we present in Chapter 6 the characterization of the examples and we investigate how much a change in coverage and/or size of the examples used to train an WFR-learning algorithm will impact the quality of the solution. We conclude in Chapter 4 with a brief list of our contributions, their limitations, and some of the future work we envision.

Chapter 2

Décrire une formulation en tant qu'énoncé ne consiste pas à analyser les rapports entre l'auteur et ce qu'il a dit (ou voulu dire, ou dit sans le vouloir); mais à déterminer quelle est la position que peut et doit occuper tout individu pour en être le sujet.

MICHEL FOUCAULT, IN
'*L'ARCHÉOLOGIE DU SAVOIR*' (1969)

State of the Art

In this thesis, we propose a thorough methodology to evaluate the inductive power conveyed by a set of application examples in order to assess the quality of the learning process in the context of MDE. In this chapter, we introduce the context of the study. As a background to understanding key concepts that are indispensable to our investigation, we briefly describe MDEs power of abstraction, how it provides adequate support for the abstract representation of systems, as well as its fundamental theory and concrete artifacts. We then describe how SBSEs power of automation helps to address the complexity of problems confronting MDE. Indeed, SBSE, by means of metaheuristics, makes it possible to solve problems of SE which until then could appear unsolvable — or whose resolution would be too complex to be automated. The reformulation of an SE problem into an SBSE problem requires two elements: the description of the structure of the potential solutions and a fitness function capable of ordering the solutions between them. Then, SBSE explores the solution space by successive refinements of an initial population and evaluates candidate solutions by means of the fitness function. In more than two decades, the application of GP has proved its worth, as shown by the plethora of studies identified by the SBSE repository (Zhang et al, 2014).

However, research on SE automation went further. Using application examples to capture the semantics of a specific task, a semantic guidance can be incorporated into the SBSE search mechanism. In the last years, Example-Based automation has reached a certain maturity. Notwithstanding the potential of EB-MDE technics, we introduce how related works suffer from a fundamental limitation. The point that we would like to stress here is the dependence on the *quality* of examples that such a process features. If the examples are *representative*, the solution will be *good*. Yet there exists no consensus on the measurement of representativeness of examples. After a brief overview of the background, we introduce work on EB-MDE as well as model generation since they are directly related to our investigation.

In short, this chapter is structured as follow. Section 1.1 presents key principles and artifacts of MDE. Section 1.2 introduces SBSE and how the reformulation of SE problems into optimization problems helps MDE automation. This section as well shows how learning from examples adapts well to SBSE reformulation. Section 2.1 introduces Example-Base Model-Driven Engineering through different concrete examples of application. Finally, Section 2.2 draws the state of the art of automatic model generation techniques.

1. Background

1.1. Model Driven Engineering

Models have played a secondary role in Software Engineering until the advent of Model Driven Engineering (MDE). The benefits of using models in Software Engineering has proven to be greater than in any other engineering discipline (Selic, 2003). Using models fosters the overall quality of the software product since changes in requirements become changes in the domain model; it is thus easier and faster to maintain the system (Selic, 2003; Mohagheghi et al, 2013).

More precisely, MDE assumes that employing domain concepts rather than code implementation concepts makes the modelling of complex systems easier. Still, a software system actually operates through its implementation artifacts. That implies that starting from domain models, a mechanism is needed to produce, modify, and test low level artifacts. Indeed, Sendall and Kozaczynski (2003) have shown as early as 2003, that from a high-level model of a system, the underlying code that will effectively be run can be generated automatically. Moreover, models are subject to modification and evolution through the software lifecycle.

Manually modifying an artefact and trying to keep others up to date has proven to be an arduous task that is often disregarded and prone to errors. Automatic mechanisms are emerging with works on the co-evolution of metamodel and model (Kessentini et al, 2016), and semiautomatic mechanisms for the co-evolution of wellformedness rules and metamodel (Batot et al, 2017). Yet, more work is needed to cope with the discrepancy generated by the development process (Paige et al, 2016).

1.1.1. *Metamodelling*

Models in MDE come with a precise definition of their structure. This definition, also called *modelling space* or *metamodel*, allows models to be part of automated processes. Concisely, a metamodel is a model that specifies concepts that form part of a modelling language, with their possible interrelations and the constraints they must satisfy. *Models* are instances that conform to the metamodels specifications. A metamodel is thus a model that describes models at the lower abstraction level.

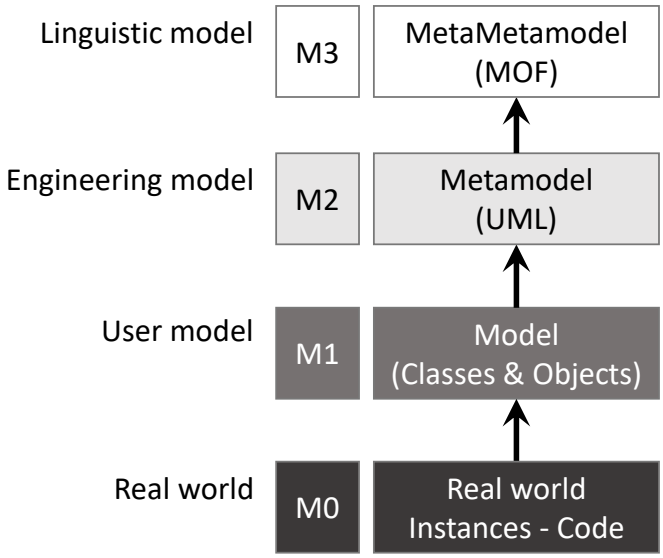


Figure 2.1. OMG structure with four abstraction levels

The Object Management Group (OMG) has defined a structure in four levels of abstraction that aligns with the principles of MDE called *MetaObject Facility* (MOF[1]). Levels are illustrated in Fig. 2.1. First, a metamodelling language (at the linguistic level M3) specifies

[1]https://www.omg.org/mof/

the structure of metamodels at the engineering level (M2). A metamodel defines the language structure used to represent the instances of a specific domain. A metamodel is also called Domain Specific Language (DSL). As an example for M2 level metamodel, UML is the *defacto* language for software modelling. Then, the level M1 contains instances conforming to the metamodel described in M2 or *models*. An instance of UML will be a diagram - for example class diagram, activity diagram, sequence diagram. Finally, M0 is the object level, the real world of which diagrams are representations.

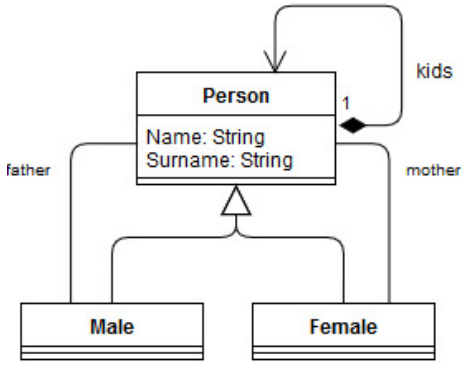


Figure 2.2. Structure of the `FamilyTree` metamodel

As an illustrative example, Fig. 2.2 depicts a simple metamodel to represent family trees. A `Person` has two attributes: its `name` and `surname`; and three references. References `mother` refers to a meta-class `Female` and `father` to a meta-class `Male`. The reference `kids` is a composition, which means that if the instance of a `Person` is deleted, its referred `kids` will be removed from the representation as well. Attributes and references are also called features, or meta-features. The empty arrows specify that `Male` and `Female` are specializations of the concept `Person` with specific characteristics.

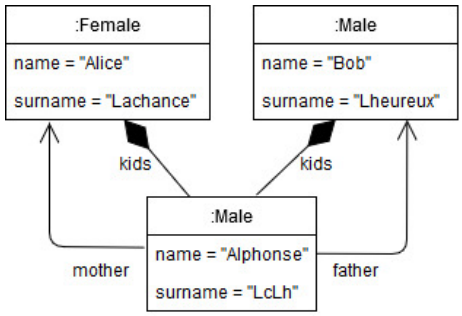


Figure 2.3. Instance conforming to `FamilyTree`metamodel

Fig. 2.3 illustrates an instance that conforms to the metamodel `FamilyTree` shown in Fig. 2.2. It contains three Persons, two Male, one Female. Two of them have a kid, which is a Male, and whose name is Alphonse. Alphonse has Alice as mother and Bob as father.

More pragmatically, the *MetaObject Facility* (MOF) is the foundation of [Object Management Group] OMGs industry-standard environment where models can be exported from one application, imported into another, transported across a network, stored in a repository and then retrieved, rendered into different formats (including XMI, OMG's XML-based standard format for model transmission and storage), transformed, and used to generate application code (OMG, 2013). MOF benefits the larger and most active modelling community. The Eclipse Modelling Framework (EMF) is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model. Its core is described by Ecore metamodel, the *defacto* reference implementation of OMG's EMOF (Steinberg et al, 2009).

1.1.2. *Precise metamodeling*

We have shown previously how to define a metamodel using MOF. However, MOF does not enable the expression of constraints relating to the use of a metamodel in a particular domain. Moreover, some hidden contracts, self-evident to an expert but impossible to express with MOF, might need be considered. To express concise well-formedness rules, another language must be used. The Object Constraint Language (OCL) (OMG, 2012) makes it possible to define at the metamodel level pre- and post-conditions on operations as well as invariants on attributes and references.

Let us illustrate our words with a simple example. In our `FamilyTree` case, nothing prevents us from representing a Person which mother is itself. Listing 2.1 expresses this hidden contract in OCL. Literally, for all and every `Person` (the metaclass context) an

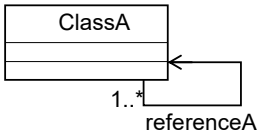
Listing 2.1. Wellformedness rule 'not-own-mother/father'

```
Context Person:
  inv not-own-mother:
self.mother <> self
  inv not-own-father:
    self.father <> self
```

instances mother (`self.mother`) must be different from itself (`self`). Keyword `inv` stands for invariant, and `not-own-mother[father]` are invariant names.

If OCL is very easy to understand with small, legible and clear examples, this is not the case for larger instances. As a matter of fact, manual writing of OCL constraints turns out to be arduous, and error prone. To alleviate this issue, Cadavid Gómez (2012) dedicated his thesis to an empirical investigation of more than 400 metamodel applications from industry and academe alike. He found that 21 patterns emerge from the use of OCL. 21 patterns which, composed together with logical operands, permitted the expression of all useful constraints *i.e.,* with respect to the 400 cases at hand. More precisely, 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 description given in Fig. 2.4 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 reference the operation closure, which can only be invoked on collections.



- **OCL Expression Template:**

```
context ClassA inv AcyclicReference : referenceA
  ->closure(iterator: ClassA | iterator.referenceA)
  ->excludes( self )
```

- **Parameters:** ClassA, referenceA

Figure 2.4. An OCL pattern definition: the AcyclicReference pattern

As an illustrative example, Fig. 2.5 shows a generalization of the constraint in Listing 2.1 that involves two instances of a pattern, composed with a logical *AND*. This constraint avoids having instances with loops in the ancestry of a `Person`.

Conclusion: As a take away, we consider that using OCL patterns reduces the solution-space dimensions. Instead of considering all possible instances that OCL allows to write, the granularity is set by the rigid representation we decribed: a tree-like structure whose

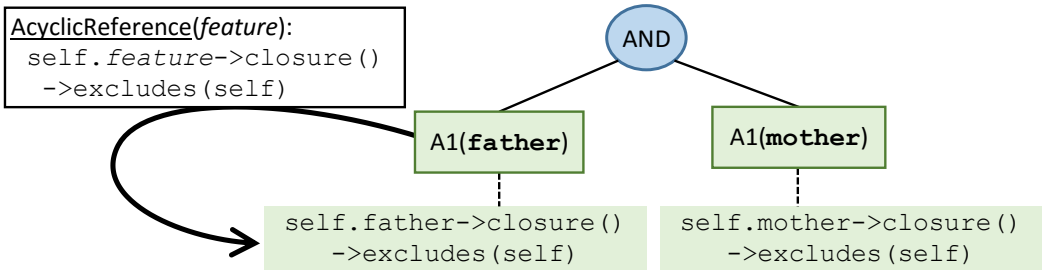


Figure 2.5. Abstract syntax tree of Wellformedness rule 'ancestry-bares-no-loop' with instantiations of pattern `AcyclicReference`

nodes are logical operators, and whose leaves are instances of OCL patterns. We call the resulting solution space *pragmatic space* since, if its coverage is equivalent to the solution space (Cadavid Gómez, 2012), its exploration is many times more efficient.

OCL patterns act mainly on the structure of metamodels. Specific knowledge such as thresholds and rules on literals are little chance to be considered since they are specific to the targeted application domain. This (more general) problem is illustrated by Clarisó and Cabot (2018) where authors show how much an iterative process can address the issue. A consequence to that statement is that a human factor is indeed essential to supervise the edification of a precise metamodel definition.

1.1.3. *Model Transformation*

In previous sections, we introduced models, metamodels, and the task of validation with well-formedness rules (OCL constraints). Another key artifact to MDE is automatic model transformation, or model transformation (MT). A model transformation can be defined as the automatic generation of a target model from a source model, according to a set of transformation rules where each rule analyzes some aspects of the source model given as input and synthesizes the corresponding target model as output. Both source and target models conform to their respective metamodels (Kleppe et al, 2003). Applying a MT literally *transforms* a model into another one. Fig. 2.6 shows the structure of a model transformation. The transformation itself is written in a language which metamodel is clearly defined. It can be a programming language such as Java, or C, or a dedicated language such as ATL[2] or QVT[3].

[2] `http://www.eclipse.org/atl/`
[3] `https://www.omg.org/spec/QVT`

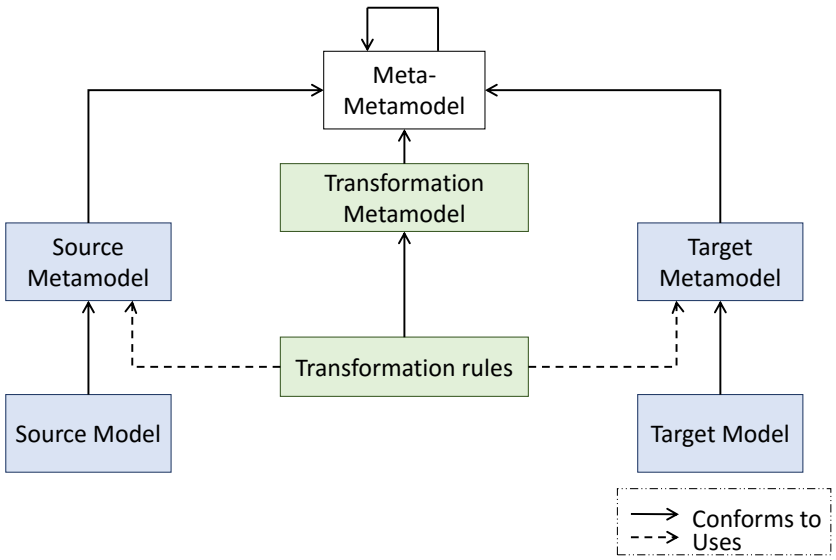


Figure 2.6. Structure of a model transformation

Transformations with source and target models that conform to the same metamodel are said to be endogenous, whereas when they conform to two different metamodels, transformations are said to be exogenous. Exogenous transformations can be separated into vertical and horizontal. A vertical transformation has source and target models at different abstraction levels. This is the case for model-to-text transformation, for example, or for code generation (M2 to M1), as well as for the derivation of concepts from instances. A horizontal transformation has source and target models at the same level of abstraction. For more details on model transformations, we advise the reader to look at the taxonomy proposed by Mens and Van Gorp (2006), and the in-depth feature-based classification by Czarnecki and Helsen (2006).

1.2. Search Based Software Engineering

Search Based Software Engineering (SBSE) advocates for the reformulation of complex SE problems into optimisation problems. To solve these problems, SBSE uses methods from a subfield of stochastic optimization (*i.e.,* field of optimization where problem data are uncertain and modeled through a probabilistic distribution) known as metaheuristics (Luke, 2013).

1.2.1. *Metaheuristics*

Luke (2013) argues that metaheuristic algorithms are used to find answers to problems when you have very little to help you: you do not know beforehand what the optimal solution looks like, you do not know how to go about finding it in a principled way, you have very little heuristic information to go on, and brute-force search is out of the question because the search space is too large. Nonetheless, if you are given a candidate solution to your problem, you can test it and assess how good it is. In short, Search Based Software Engineering (SBSE) builds on the assumption that it is possible and profitable to reformulate some SE problems into optimization problems where traditional deterministic automatization technics fail. The general structure of SBSE methodology is schematized in Fig. 2.7.

Representation and fitness function. To reformulate an SE problem into an optimization problem, two concepts must be redefined (Harman and Jones, 2001). The first is a *representation* of the solution space — *i.e.,* a description of the structure of candidate solutions. That description comes with the redefinition of genetic operators such as crossover and mutations that will stir the solutions genotype (syntactic material) during evolution. The second key concepts to SBSE that must be redefined is the *fitness function*. A fitness function is a mechanism able to distinguish which of two solutions performs better. This is in order to select candidate procreators to produce the next generation. A termination criterion will decide the characteristics a solution must reach to end the evolution. At the end of the day, successive generations of solutions will produce near optimal solutions, *i.e.,* solutions that fall within a specified acceptable tolerance.

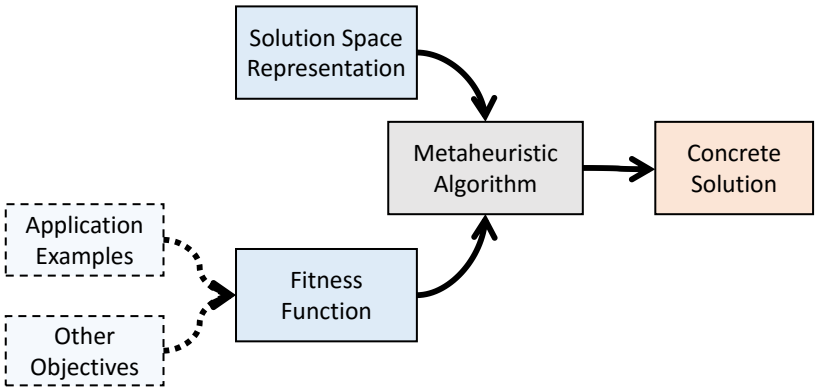


Figure 2.7. Structure of Search-Based Software Engineering

The difficulty of establishing meaningful representation and fitness function is not addressed here. We will, however, briefly discuss briefly principles that may be useful for the understanding of this thesis. First, it is important to define a search space whose variations of topology are valued by the fitness function. If the changes from one individual to another are too abrupt, peaks will take shape in the research landscape and local optima may trick the search. The evolution may miss potentially better solutions. On the other hand, if changes from one individual to another are too small, generations may explore many very similar solutions. A very detailed but slow exploration of the landscape will depend heavily on the quality of the original population. That being said however, the fitness function must define a search landscape whose modularity can be strategically explored by generations of solutions without the risk of falling into some local optimum.

Overall, growing numbers of authors and domains of application assess the accessibility and usability of the method. A good literature on both fundamentals and applications of SBSE is found in the literature review by Boussaïd et al (2017).

1.2.2. *Computation/Algorithms*

Many different metaheuristic algorithms exist. Boussaid makes a fundamental distinction between two main classes of methods: Single-state and Population-based metaheuristics (Boussaïd et al, 2013).

Single state methods. Starting from a given solution, the algorithm will apply changes, or mutations, describing a trajectory into the search space. One or more neighborhood structures must be defined. Arguably one of the simplest metaheuristics, single state technique, is the process of examining neighboring solutions for a fitter candidate, and once found the fitter candidate becomes the current solution (Clarke et al, 2003; Luke, 2013). The process is repeated until an *acceptable* solution is found.

A popular single state method is simulated annealing, a technique inspired by the annealing process of metals used in metallurgy. A crystalline solid is heated and then cooled according to a controlled cooling until it reaches its most regular crystalline configuration possible. To achieve this, the energy state of the network is minimized, thereby obtaining a homogeneous material structure. Homogeneity is measured with the fitness function.

Population-based methods. As opposed to single-solution methods, Population based methods start from a set of solutions (*i.e.,* a population). One approach, called Evolutionary Computation (EC) (Fogel, 1999) or Genetic Programming (GP) (Koza, 1992), consists of a mimic of Darwins theory of evolution On the Origin of Species by Means of Natural Selection (Darwin, 1859).

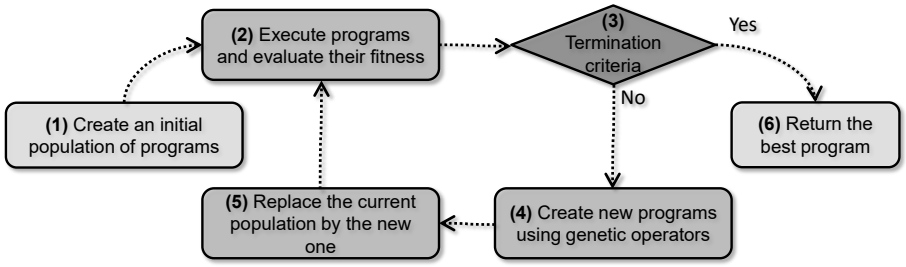


Figure 2.8. A classical GP run

Concretely, a first population of N solutions is created, and its individuals are bred until certain characteristics arise. The breeding can be an endogenous modification of an individual such as a mutation in the case of single-state methods; or an exogenous modification applied across two individuals in the case of Population-based methods. The result is an improvement in the capacity of individuals against a well-defined fitness function whose main goal is to define the chance that an individual has to take part in the production of the next generation. More technically, the easiest way to understand GP is to look at Fig. 2.8. The first step is to create an initial population (of one or more individuals). Then, these individuals are evaluated and, if the termination criterion is not reached, new individuals are created by mutation and/or sexual reproduction. Newly created solutions are then evaluated using the fitness function. The process is repeated until the termination criterion is reached. The main advantage of GP in SE is that it allows solution candidates to be complex executable artifacts. Once a new solution is created, it can be executed to evaluate its performance.

Another Population-Based method is Swarm Intelligence (SI). Here, the knowledge being learnt is not contained in a single solution (part of the population) but consists of a combination of individuals solving each a part of the problem.

The choice between the different methods depends on the structure defined for the solution space and on the designer's sensitivity. Luke (2013) gives extensive details on each and every algorithm, including pros and cons, in a remarkable literature review addressed to undergraduate teaching. Rather than the kind of algorithm to use, what concerns us in

this thesis is the way in which all these algorithms use the fitness function. More precisely, we are interested in the opportunity offered by MDE to capture the expected semantics of a solution through application examples. We will see in more details Example-Based MDE in Section 2.1.

1.2.3. *Multi-Objective Optimization Problem*

We must add a few words about the design of the fitness function. As we briefly mentioned earlier, the fitness function tells us how good a solution is. It is an evaluation which is able to distinguish between two solutions which one is better. However, – due to the high level of complexity of the problem intended to be solved, this evaluation can rarely be coined into one single objective. Furthermore, although it is sometimes helpful to combine different criteria into a single objective, high-level criteria are generally contradictory in nature (*e.g.,* size and coverage, cost and energetic efficiency) and should be considered apart from one another. From this arises Multi-Objective Optimization Problem (MOOP). To address this difficulty, Pareto dominance is a generally well accepted solution. In a nutshell, a solution is Pareto optimal if no feasible solution exists which would improve some objective without causing a simultaneous worsening in at least another one.

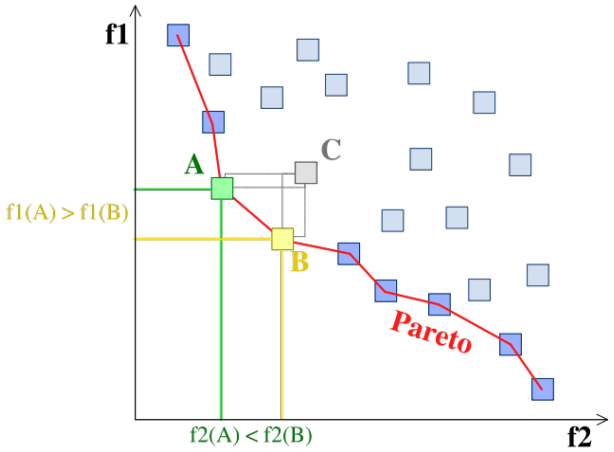


Figure 2.9. Pareto ranks

The rationale is straight forward: Individual A Pareto dominates individual B if A is at least as good as B in every objective and better than B in at least one objective. A Pareto rank is a set of solutions that are not dominating each other. The Pareto-optimal Front contains solutions that do not dominate each others but dominate all other solutions. An illustration

of Pareto Front using two objectives is found in Fig. 2.9. An optimal solution would reach the bottom left corner. Deb et al (2000) proposes a Non-Sorting Genetic Algorithm (NSGA-II) to solve MOOPs. It is now widely used by the SBSE community.

It is important to note that recent research in GP emphasizes that using semantic objectives – *i.e.,* objectives relating to the semantics of solutions while applying genetic operations acting on their syntax, speeds up evolution and produces more consistent results (Vanneschi et al, 2014).

Our second contribution is an improvement of GP algorithms in favor of a diversity relating to social consideration of individuals in a population of solutions. The details of the evolutionary algorithm mechanisms, as well as the rationale behind social semantics and its implementation, are depicted in Chapter 5.

2. Related work

2.1. Example Based Model-Driven Engineering

A few years ago, Bąk et al (2013) coined the term Example Driven Modeling to refer to works focusing on the use of examples during the process of modeling. Authors promote the use of examples to foster understanding of MDE artifacts: models, metamodels, and transformations. Examples (*i.e.,* models capturing a specific semantics, or structure, useful to the understanding of the application domain), coupled with their abstract definition, are indeed precious communication tools that quicken not only inter-modellers exchange, but modellers-stakeholders exchange as well (Bąk et al, 2013). This work finds its root in the idea of a generative perspective on programming (Czarnecki et al, 1997).

If the foundation of SBSE posits that it is easier to say if a solution is good or not than to actually build a good solution, **Example-Based automation of MDE (EB-MDE) stands that it is easier to provide examples of applications that illustrate a problem well than to draw a general theory about how to solve it.** Indeed, in the last decade, researchers have shown that it is feasible to feed metaheuristic algorithms with application examples.

2.1.1. *Learning from examples*

We have seen that MDE gives way to capture the semantics of the problem with examples of application, and SBSE offers mechanisms to support the automatic search of solutions in a multi dimensional space. In addition, MOOPs enables the consideration of objectives of a completely different nature during the same search. Thus, an objective can focus on the semantics a solution must convey to be acceptable; while the other can characterize its syntax to avoid, for example, solutions from growing uselessly big. When evaluating a solution, one objective measures the rate of examples accurately processed, and the other objective keeps its size as small as can be, or any other syntactic needs.

For that matter, MDE looks like a perfect candidate to use SBSE technics to automate (or learn) complex artifacts. The semantics captured in the examples can be used as a search objective to evaluate a solution relevance – *i.e.,* the success rate of the solution when executed on the set of examples. Artifacts are thus learned automatically from examples.

In the case of automatic learning of MDE artifacts, an input and its corresponding output after execution must be provided to form one example of application. **An example is thus formed by two models: one input and its corresponding output**. Their metamodels describe potential input and output data (or problem and solution spaces). Several teams have been using examples to learn high-level MDE artifacts. We give here a short list. More details will be found in Chapter 5.

2.1.2. *Learning model transformations*

Model transformation is the most studied artifact and figures as a leader in MDE learning research (Babur et al, 2018). As early as 2006, Varró (2006) proposed a semi-automatic graph-based approach to derive transformation rules using interrelated source and target models. Balogh and Varró (2009) have built an extension that derives n-m (instead of 1-1) rules using Inductive Logic Programming (ILP) and Wimmer *et al.* propose a similar approach with mappings defined in a concrete rather than an abstract syntax (Wimmer et al, 2007; Strommer et al, 2007; Strommer and Wimmer, 2008). Similarly, García-Magariño et al (2009) propose an ad-hoc algorithm to derive ATL n-m rules using example pairs with their mappings. Later on, teams started to automate the learning of model transformations by analogy. They do not try to abstract the transformation, they derive the corresponding

target model from a source model by considering model transformation as an optimization problem. The problem is addressed using particle swarm optimization (PSO), followed by a combination of PSO and simulated annealing (SA) (Kessentini et al, 2008, 2012a,b). More recently, Faunes et al (2013b) proposed an approach to learn directly the code of transformations from examples. Genetic programming (GP) is used to learn n-m transformation rules starting from source and target examples without the transformation traces. To learn a transformation, examples of models are given with their corresponding transformed version. The approach is enhanced by Baki et al (2014); Baki and Sahraoui (2016) to learn the rule execution control.

2.1.3. *Learning wellformedness rules*

As presented in Section 1.1.2, a set of WFRs defines a subspace in the modeling space. To capture the limits of this space with examples, valid models, *i.e.,* being part of the subspace, and invalid models, *i.e.,* not being part of it (yet conforming to the metamodel), must be provided. Faunes et al (2013a) shows that it is feasible to learn wellformedness rules (OCL constraints) from examples and counter examples with ultimately no extra knowledge.

Faunes approach consists in using the set of examples and counter examples to participate in the evaluation of candidate solutions during a mono-objective GP run. In this scenario, a good solution differentiates between valid and invalid models accordingly. The results are promising but the use of mono-objective seems problematic here. In fact, multi-objective GP allows dissociating between semantic and syntactic objectives. This permits the consideration of semantic diversity and thus promotes the generalization power of solutions (Vanneschi et al, 2014). Moreover, authors used oracles (*i.e.,* the expected solution) to produce examples – which is irrelevant for an execution *in vivo*.

More recently, (Dang and Cabot, 2014) proposed a framework to infer OCL invariants from examples. Their work consists of translating OCL in CSP logic and using a solver to identify candidate solutions. The originality of this work is the consultation of the user during the process of learning. The process is iterative, and users are asked to check whether examples make sense and to rebuke the extravagant ones. They are also asked to point out the specific problematic part of malformed examples when applicable to help with providing

examples in the next iteration. Later, the user is asked to assess the relevance of the solution(s) given by the algorithm. To our best knowledge, this is the first time that there is a concrete evaluation of solutions relevance.

In the same vein, Clarisó and Cabot (2018) show the complexity of expressing and repairing specific misconceptions such as thresholds and complex select expressions. The authors show that mutation analysis is a good candidate to tackle this issue.

Our third contribution shows a new version of the approach by Faunes in which we enhance the abstract structure of wellformedness rules with quantifiers and integrate our discovery about injecting a social dimension in the evolution mechanism. We also consider user potential insight with the configuration of coverage definition during model generation (using the tool described in Chapter 4). Finally, the relevance of solutions, as pointed out by Dang and Cabot, is evaluated to measure the performance of the algorithm. We do not use it as feedback, since we want to evaluate the dependence of solutions quality on the coverage and size of the examples for an automatic learning process.

2.1.4. *Other applications*

In the same vein, Mokaddem et al (2018) published a work to assist pattern recognition using examples of pattern matching, and Saied et al (2018) used traces of API executions as examples to derive temporal usage patterns. Finally, Faunes et al (2011) use examples to derive high-level abstraction from legacy software. These last studies are not directly related to our investigation since they are looking for patterns, rather than learning a concrete artifact, but the use they make of examples to lead a search in a multi-dimensional space will directly benefit from our investigation.

2.1.5. *Conclusion*

In conclusion, we want to stress a limitation for studies focusing on learning from examples. All of them use examples they either (1) build *ad-hoc* for the specific purpose of their study; or, (2) pick from the industry to show the scalability of their algorithm. BUT — there is not, to our knowledge, studies focusing on the representativeness of the examples used to train their algorithms. This is not new in MDE. Since its early days, the MDE community has been striving to address quality (Mohagheghi and Aagedal, 2007). In 2016 a special issue on quality in model-driven engineering was published in Software Quality Journal (Amaral

and Mernik, 2016). Yet, the closest work to our knowledge was done by (Fleurey et al, 2009). In this work, authors aim at qualifying input data for model transformation testing. Given the need for qualification of MDE artifacts, we envision that a characterization of examples for learning will put a corner stone to the improvement of MDE automation.

2.2. Model generation

A very convenient advantage of using well defined models is the possibility to use their metamodel to automatically generate other models. Research focusing first in test case generation led to a lot of different tools and methodologies to generate automatically instances conforming to a given metamodel (Wu et al, 2012).

2.2.1. *Model Transformation Testing*

Pragmatically, the main goal that guided research on model generation is to provide model transformation testing with input data (Fleurey et al, 2009). It is a matter of decomposing a metamodel into fragments to discretize the space it defines. Two kinds of fragments are used. Model fragments are different strategies to compose object fragments. Objects fragments are features from the metamodel with their respective different possible instantiation values. The overall idea is to measure how many of the metamodel constructs have been instantiated to form a model. The scale becomes how many of the classes and features of a metamodel need be instantiated at least one to produce a model. We call this measure the *coverage* of the model over the modelling space. The feasibility to produce perfect test suites is not sure (Baudry et al, 2006, 2010), the use of fragments nevertheless brings encouraging results for learning (Batot and Sahraoui, 2016).

Another aspect to account for during the automatic generation of models is the consideration of complex constraints. The most advanced work that we know of is Popoola et al (2016). It focuses on generating models for large and heavily constrained metamodels.

As a conclusion, research on model transformation testing led to the formulation of four essential features a model generator must have to ensure a robust and productive generation:

- **Scalability of the technique**: Can the technique be used for large size metamodels? Does it allow to generate large instances?
- **Generic approach**: How generic is the approach? What is its scope?

- **Flexibility of the application**: Does the technique take into account the well-formedness rules?
- **Diversity of solutions**: Does the method make it possible to generate various models? Does it take into account a goal of diversity between the generated models?

2.2.2. *Metamodel Instance Generation*

To our knowledge, the first work directly related to examples generation was done by Gogolla et al (2005). In this work snapshots, or object diagram instances, are generated with USE tool (Gogolla et al, 2007) to test and certify UML and OCL models. Other teams tend to generalize to a broader definition of metamodels (Brottier et al, 2006). They ground the generation on exploring the structure of the metamodel to produce compliant instances. Studies and tools come with their pros and cons and no method is unanimously accepted. The community questions the origins (and consequences) of this lack (Wu et al, 2012).

Three main trends coexist in the field:

- **Partition-based coverage criteria**, as seen for model transformation testing, consists in generating metamodel instances using criteria formulas to further constrain the entire generation process. The generation is done with the goal to *cover* at best the modelling space.
- **Graph-property based criteria** consists in generating metamodel instances by encoding graph properties into formulas according to different scenarios where the metamodel is translated into a graph (with inheritance). This is the most promising approach for the generation of one single instance of great size (*i.e.,* instead of set of instances) but the translation into formulas remains uncertain.
- Generation as **Satisfiability Modulo Theory (SMT) problem** consists in generating metamodel instances by translating a metamodel to an SMT problem via a bounded graph representation. Again, the translation into SMT is not easy – lots of skills with SMT is required. More, the use of Alloy (Jackson, 2006), on which most approaches are based on, seems problematic with complex instances and show a limited scope (Anastasakis et al, 2007; Kuhlmann and Gogolla, 2012).

Whether exploring features strategically or assisted with a constraint solver, the problem is extremely complex, and the technologies developed so far show limitations either in terms

of scalability or interoperability (Wu et al, 2012). Most of the existing work on instance generation, e.g., (Gonzalez and Cabot, 2014; Gogolla et al, 2015), target specific tasks and purposes without a generalization effort. We also note that none of the studies are concerned with providing models on the specific purpose to feed learning algorithm.

2.2.3. *Model Set Selection*

One of our most important difference when generating model sets for learning is the first-hand consideration we make of the size of the sets. Indeed, when constructing the sets, an expert must complete each partial example (input model) with its corresponding output part. It is thus necessary to keep example sets as small as possible. Our adaptation, using a multi-objective genetic programming algorithm, allows us to consider coverage and size of the sets independently.

This forms our first contribution (see Chapter 4 for details). In our approach, instead of generating directly a set of models, we implemented an approach that selects models to form a covering yet minimal set. It uses a partition-based consideration of coverage, as well as a size measurement enhanced with a choice between different minimality criteria. In the same manner, Sen et al (2009) generate hundreds of models and then select a set for model transformation testing. They use their tool Cartier to generate the models and then perform mutation analysis to select a (most-)covering set. Our approach also accounts for a specific coverage definition that the user can specify.

2.2.4. *Nota*

At the time I am writing these words, a work from Semeráth et al (2018a) came out with a new coverage measurement definition. This new measurement appears as a significant improvement in the field since it takes an in-depth consideration of the instantiation (*i.e.,* the depth and diversity of references indirection paths and attributes values). The present study does not use this new methodology due to chronical mismatch. However, we use coverage as a specific concern and assure its integration is not only possible but relatively easy. One of our future work is a reproduction of the entire study using this new measurement.

3. Summary

In this chapter, we presented how MDE benefits from the power of automation of SBSE. The former offers adequate support for describing both the problem space and the structure of the desired solution(s). The latter allows the resolution of complex problems thanks to their reformulation into optimization problems. With the combination of application examples and metaheuristic algorithms, it has been found that it is possible to automatically learn very complex MDE artifacts. The conformity of the output of the solutions with the description given by application examples provides a valuable semantic objective for the search for solutions. Studies abound in this direction and show the merits and potential of the method.

However, it seems clear to us that a lack of representativeness in the examples will impact the quality of the artifact learned. Yet, to our knowledge there has been little work done to highlight and quantify this relationship. This observation is the basis of our research contribution. Indeed, we believe that a qualification of example sets based on their representativity of the problem is essential to the assessment of learning technologies and furthermore to the automation and adoption of MDE.

Chapter 3

'The forest will swallow you.'
'Then I will become a tree,' I said.
'Then they will cut you down because of the road.'
'Then I will turn into the road.'
'Cars will ride on you, cows will shit on you, people will perform sacrifices on your face.'
'And I will cry at night. And then people will remember the forest.'

BEN OKRI, IN
'*THE FAMISHED ROAD*' (1991)

Research map

The objective of this chapter is to draw a big picture of our investigation. In the previous chapter, we paid special attention to Example-Based learning approaches and to the advancement of automatic model generation. In following sections, we introduce the general research map of our investigation before we go into detail about how our approach, at the very intersection of these two fields, contributes in **improving Example-Based automation in Model Driven Engineering (EB-MDE)**.

1. Thorough learning process

Faunes Carvallo (2012), with his colleagues, did a milestone work proving that SBSE offers an ideal support to automate the learning of MDE artifacts from examples (or By Example). His study addresses the learning of wellformedness rules (WFR) (Faunes et al, 2013a) and model transformations (MT) (Faunes et al, 2013b), as well as the derivation of concepts from legacy code (Faunes et al, 2011). Since then, studies on MT learning have flourished and the overall process has matured significantly, as shown in Chapter 2, Section 2.1. However, during the same period, only a few attempted to automate WFR inference from

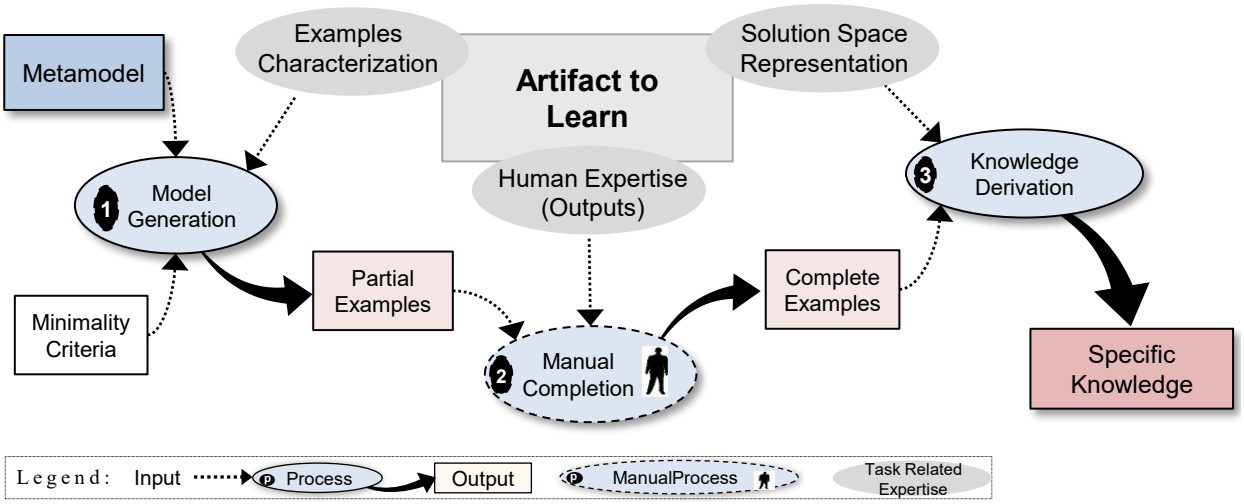


Figure 3.1. Research map: a thorough process for learning artifacts in MDE

examples. Notwithstanding that difference in maturity, a common trend emerges from the joint consideration of the different techniques. More generally speaking, we see a recurring pattern in the process of most of the learning approaches we found. As depicted in Fig. 3.1, the thorough learning process presents three main steps that exploit different facets of the task or artefact to learn. At the very beginning (1), a set of partial examples must be provided. Using a metamodel description, together with some generic criteria such as coverage and size, enables the automatic generation of model sets. Then, each example must be completed manually (2). This part *cannot* be automated since completion comes from the expertise that an expert of the domain establishes. Finally, with the assistance of a *solution space representation* that defines the structure of candidate solutions, a learning algorithm derives automatically the specific knowledge from the example set (3).

This high-level formalization exhibits one of the limitations that the derivation of knowledge from examples suffers from: a clear over-dependence on the quality of the examples used. A *quality* that has not yet been characterized and suffers from a lack of consideration as a field of investigation of its own. Indeed, if the relation between the representativeness of the examples used to train a learning algorithm, and the power of generalization of the solution it yields seems obvious to the community, attempts to qualify this relation remain scarce. We saw in Chapter 2, Section 2.2.2 how current work on model generation can help to address this issue.

2. Partial example generation

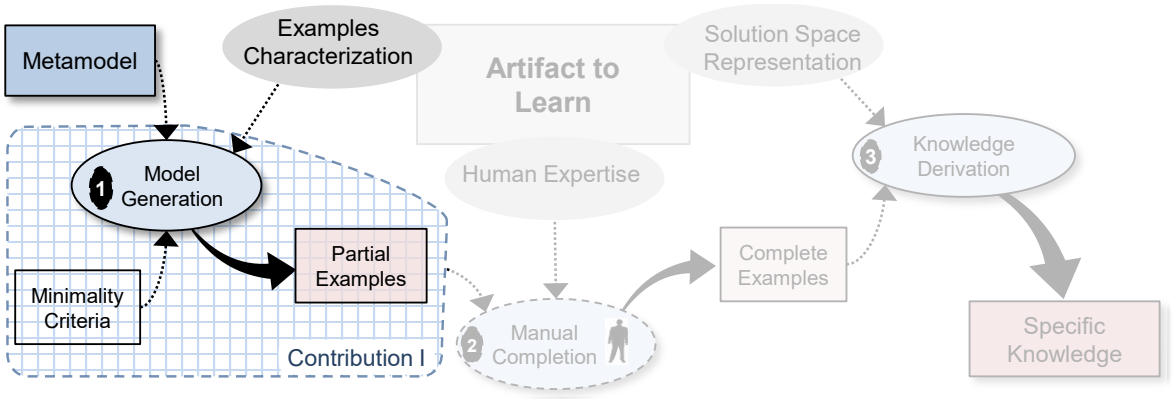


Figure 3.2. First contribution: a tailored model generation

The first detail we want to point out in the learning process is the importance of disentangling the solution space from the problem space of the task to learn. We mentioned in Chapter Ô2, Section 2.1.1 that application examples in MDE are generally composed of two models: one represents a potential input of the task and the other is the corresponding output after execution of the task. We give examples of application in Chapter 4, Section 2.1. This distinction allows **narrowing the concept of representativeness to the input space only** as highlighted in Fig. 3.2. Thence, representativeness of example sets is measured on the input part of the examples only. We call this input part a *partial example* and formalize the rationale behind in Chapter 6, Section 2.2.

In a MDE context, measuring the representativeness of models has been investigated in terms of quality of input data for model transformation testing (Fleurey et al, 2009). The rationale is to ensure that models cover at best the input space described by the input metamodel (more details in Chapter 4, Section 3.2.2). We investigated first the state-of-the-art contributions in terms of model generation and proposed a contribution to address current limitations and to adapt the metamodel instance generation to example set generation. The main distinction we make resides in the use of a mechanism to concentrate the focus of the algorithm on the part of the metamodel that matters specifically for the artifact targeted. Indeed, if coverage is commonly used to assess representativeness of models, covering the whole modelling space might be unnecessary when learning a specific artifact. If extra knowledge is available (before hand, or from previous experiments) **we offer to customize the generation with specifications about the coverage criterion.** A *coverage definition* is,

in this case, a selection of metamodel elements that shall (not) be considered during the measurement of coverage. More details are found in Chapter 4, Section 3.1.1. In addition, we distinguish our work from others with the consideration of the solutions size. Since an expert must later manually complete the partial examples, their number must be kept as small as possible. In this context, our first contribution fosters the generation of examples. It is a framework for model set selection tailored to the MDE learning process that provides ready-to-complete partial examples.

3. Specific knowledge derivation

The last step, once partial examples have been completed with their corresponding output part, is to derive the specific knowledge they embody. As mentioned in Section 2.1.1, metaheuristics in general and GP in particular offer an effective support for this step and EB-MDE has gathered considerable attention on the matter. Yet, if work on MT learning is mature, WFR learning remains slightly behind.

Indeed, Faunes et al (2013a) have proven that automatically learning WFRs from examples is feasible, however their tool suffers some serious limitations. Their approach was capable of learning complex sets of rules made of a logical composition of OCL patterns. Nonetheless, instead of searching for all possible OCL constraints, they focused only on instances of patterns dealing with reference cardinalities.

Clarisó and Cabot (2018) investigate the difficulty to express and repair specific knowledge in WFRs. They propose an iterative elicitation to address specific thresholds and complex select expressions that OCL allows to express. Yet, metamodel definitions hold functional and structural characteristics or semantics that do not depend directly to the specific domain of application. In our study, we target the generic nature of metamodel with WFR referring its structure.

More importantly, Faunes et al (2013a) used the expected solution (a set of OCL constraints) to generate the examples that were intended to validate their approach. For each constraint, they generated two models using Alloy solver: one model validating the constraint and the other invalidating it. In addition to the number of models required (60 in their case), we notice a bias, since part of the knowledge of the solution is used to construct the input. It is acceptable for a proof of concept, but ignores that this knowledge is impossible to get

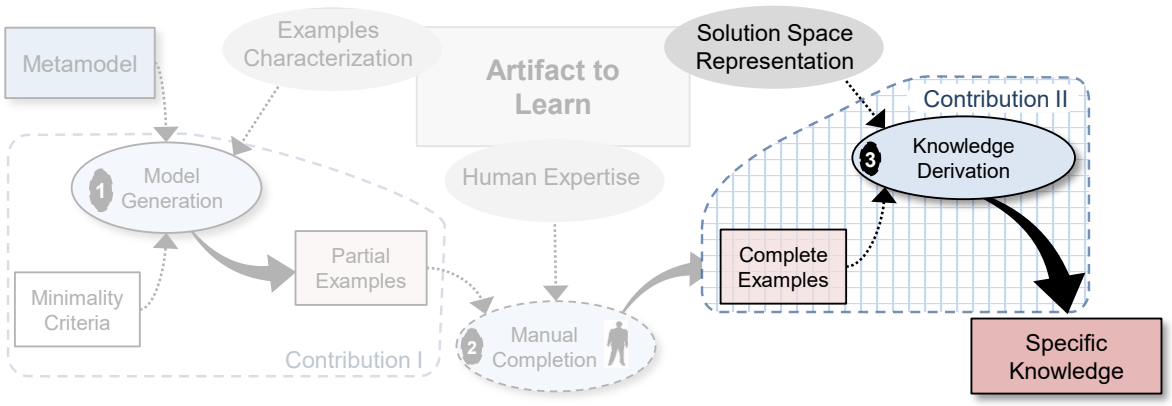


Figure 3.3. Second contribution: a significant improvement in GP

in vivo. Yet, this early work and specifically its threat to validity provoked our interest about the importance of diversity not only among examples, but as well among solutions. If promoting diversity using extra-knowledge as Faunes did, is not an option, diversity remains of importance in GP.

Indeed, Chapter 2, Section 1.2.3 mentions that a semantic diversity of solutions is crucial to avoid GP algorithms to fall in local optima. Assessing diversity is arduous, as shown by the increasing interest in SE (Wyns et al, 2006; Burke et al, 2004), and especially in MDE (Galinier et al, 2016; Ferdjoukh et al, 2017). Already Deb et al (2000) dedicated a section of his extraordinary article to draw attention to the importance of maintaining a diverse population of solutions during evolutionary computation. Later, authors have been calling for Indirect Semantic methods to optimize metaheuristic efficiency. Indirect Semantics methods "act on the syntax of the individuals and rely on survival criteria to indirectly promote a semantic behavior (Vanneschi et al, 2014). In response, we integrated an intelligent consideration of a new kind of diversity. To foster diversity during the execution of Example-Based algorithms, we employed a facsimile of a well known Information Retrieval numeral statistics (Sparck Jones, 1988) that reflects how important a word is to a document in a collection. In our case, it is about measuring how important is *an example* to a *solutions fitness* in a *population*. The rationale behind is to adjust for the fact that some examples are more frequently solved than others. In a nutshell, each example participation in the fitness of a solution is weighted by the number of solutions processing it correctly. Our Social Semantic Diversity measurement (SSD) is dynamic, since it depends on and evolves according to the

configuration of the population. Detailes are found in Chapter 5, Section 3. As we will see in Chapter 5, Section 5, using SSD speeds up the computation and fosters the generalization power of the solutions found. Full details of our implementation (located at the end tip of the thorough process in Fig. 3.3) are provided in Chapter 6, Section 3.4.2.

4. Example set characterization

Now that we shed light on the two automation spots of the learning process, lets go back to the general picture to present our approach to evaluate the relation between the coverage and size of examples and the quality of solutions. To investigate this relation, we conducted an empirical study following the thorough learning process as presented in the beginning of this chapter. Fig. 3.4 recalls this process to the readers mind and draws the relation between examples characteristics and the quality of the specific knowledge derived.

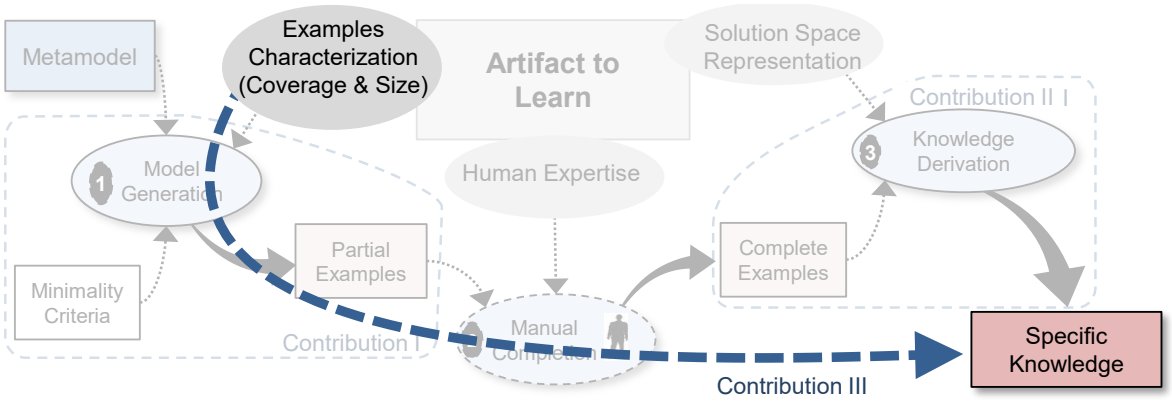


Figure 3.4. Third contribution: a characterization of training examples

To illustrate our approach, we consider the case of Wellformedness Rules (WFR) derivation from valid and invalid instances in an MDE context. In this case, the specific knowledge of the task is its ability to differentiate between valid and invalid instances accordingly to the examples.

We generated sets of partial examples automatically. They are characterized by their size and coverage. The size, as imprecise as this measure may appear, is a first step in the measurement theory and figures a solid starting point for the evaluation of complexity of SE artifacts in general (Fenton and Pfleeger, 1998). We are also interested about a size measurement since we want to account for the implication of human actors in the process. The coverage, for its part, is the *de facto* candidate to address the measurement of model sets

representativeness. The community struggles to find a tangible and consensually accepted definition for coverage. It is still under scrutiny and recent discoveries show a (re)increasing interest on the matter (Dang and Cabot, 2014). We propose, as a third contribution, a perspective on the learning process that highlights the importance of size and coverage and enables the evaluation of their impact on solutions accuracy.

We performed an empirical analysis of the relation between size and coverage of partial examples and the quality of the resulting OCL constraints. The evaluation is performed in a semi-real environment in which we know a priori the well formedness rules sought (OCL constraints provided with the metamodels). These oracles will be used to complete examples with their output parts. Oracles figures as well a *ground truth* (or oracle) that enables measuring the quality of solutions yielded by the algorithm.

The treatment of our experiment consists of deriving knowledge from example sets by means of metaheuristic algorithm. We ran multi-objective genetic algorithm NSGA-II with the optimization of the diversity of solutions mentioned in previous section.

We define the quality of solutions, our dependent variable, at three different levels. The first is a black box that checks whether or not a solution differentiates accordingly a set of examples. We measure accuracy on examples that were not used during training to try their generalisability as well. The second is an automatic measurement of the distance between a solution and the ground truth. To ensure a set of constraints is close to another one is to check if they both manipulate the same elements in the metamodel. We verify that a solution with a good accuracy manipulates the same elements as the oracle. We finally perform a manual in-depth comparison between solutions and the ground truth to ensure that a strong fitness actually accounts for relevant solutions.

Results show how subordinate the generalizability of solutions is to the representativeness of examples. These encouraging results let us envision a deeper investigation of size and accuracy attributes. They also show the numerous factors learning involves and how much work remains to be done in order to measure precisely the quality of artifacts learned automatically from examples.

Chapter 4

A Generic Framework for Model-Set Selection for the Unification of Testing and Learning MDE Tasks[1]

Edouard Batot and Houari Sahraoui
DIRO, Université de Montréal

[1]The content of this chapter has been published in ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS), October 2-7, 2016, Saint-Malo, France. Reference (Batot and Sahraoui, 2016)

Résumé. Nous proposons un cadre générique pour la sélection d'ensembles de modèles pour l'apprentissage ou le test de tâches de l'Ingénierie Dirigée par les Modèles. Nous ciblons spécifiquement les tâches qui s'appliquent à des modèles ou les manipulent, telles que la définition de modèle, la vérification de la validité du modèle et la transformation de modèle. Dans notre cadre, nous considérons la sélection d'ensemble de modèles comme un problème d'optimisation multi-objectif. Le cadre peut être adapté à l'apprentissage ou au test d'une tâche spécifique en exprimant d'abord le critère de couverture, qui sera utilisé comme premier objectif d'optimisation. La couverture est exprimée en marquant le sous-ensemble d'éléments du métamodèle d'entrée qui sont pertinents pour la tâche considérée. Ensuite, un ou plusieurs critères de minimalité sont sélectionnés en tant qu'objectifs d'optimisation supplémentaires. Nous illustrons l'utilisation de notre framework avec les tests de métamodèles. Cette étude de cas montre que l'approche multi-objective donne de meilleurs résultats que les sélections aléatoires et mono-objectives.

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 (Schmidt, 2006). 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 (Wimmer et al, 2007; Bąk et al, 2013; Cadavid et al, 2012; Faunes et al, 2013a,b; Kessentini et al, 2011b). 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, (Sadilek and Weißleder, 2008; Cadavid et al, 2012) for metamodel testing, (Brottier et al, 2006; Fleurey et al, 2009; Mottu et al, 2012; Sen et al, 2009) for transformation testing, (Cuadrado et al, 2012) 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 (Sadilek and Weißleder, 2008). 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 of 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.

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 (López-Fernández et al, 2013). An example for a metamodeling task is composed of a model, which conforms 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 (Cuadrado et al, 2012). 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 (López-Fernández et al, 2013). 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 (Faunes et al, 2013a). 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 (Brambilla et al, 2012). 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 (Baudry et al, 2010), or more generally, outputs that conform to a given oracle (Kessentini et al, 2011a). A transformation program can be learned from a set of *input models* and their corresponding output model (Wimmer et al, 2007; Baki and Sahraoui,

2016). Hence, a case (example) for testing (learning) a model transformation is composed of an *input model* and, for instance, its associated expected output model (Baki et al, 2014). 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 (Jeanneret et al, 2011) and (Mottu et al, 2012). 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 (Utting et al, 2012). 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.*

3. Framework

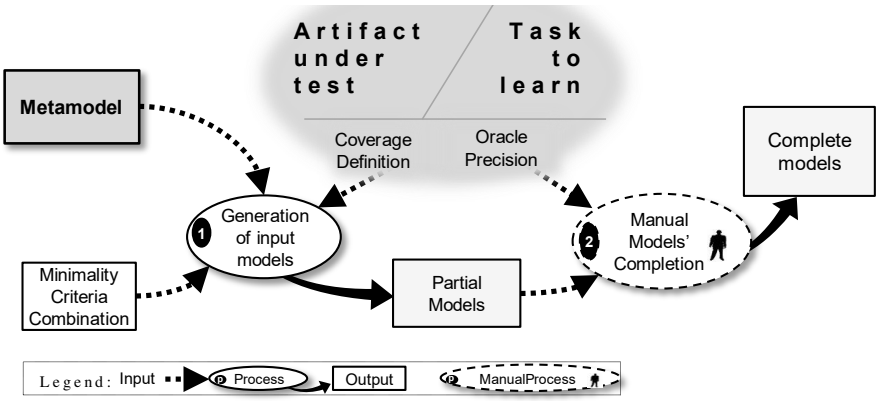


Figure 4.1. A generic framework for testing MDE activities

As stated in Section 2, MDE tasks diverge on their coverage definition and in the type of expected outputs. Fig. 4.1 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.

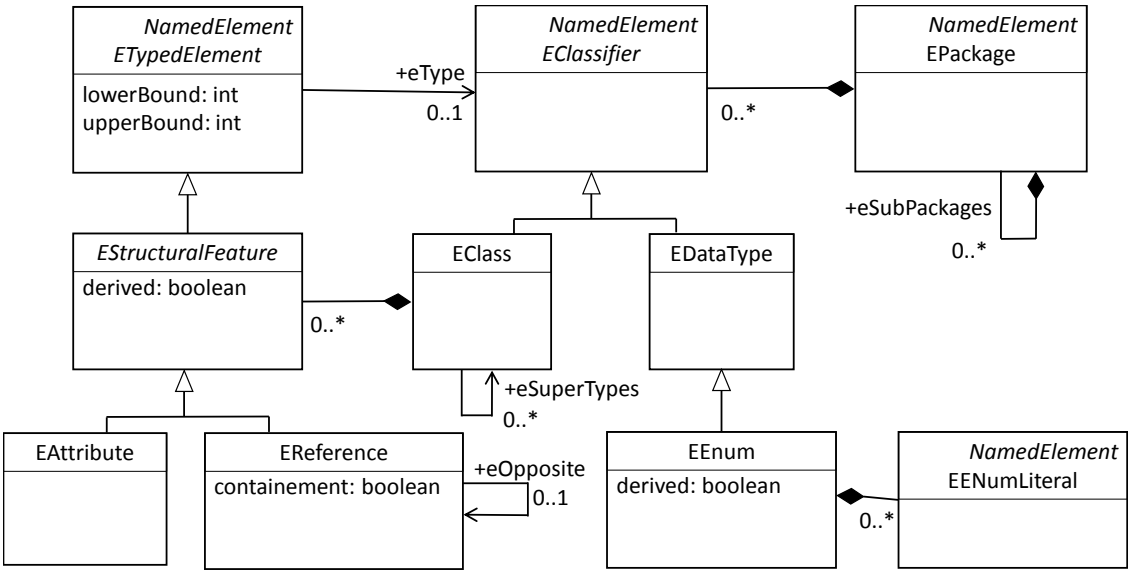


Figure 4.2. Ecore metamodel

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. 4.2). An intuitive approach to define the coverage is to consider the whole Ecore metamodel. However, Cadavid, after studying dozens of metamodels, shows in (Cadavid Gómez, 2012) that there are 21 OCL patterns that are 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 (Faunes et al, 2013a).

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

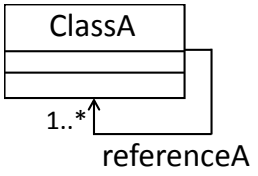


Figure 4.3. MOF structure of pattern *AcyclicReference*

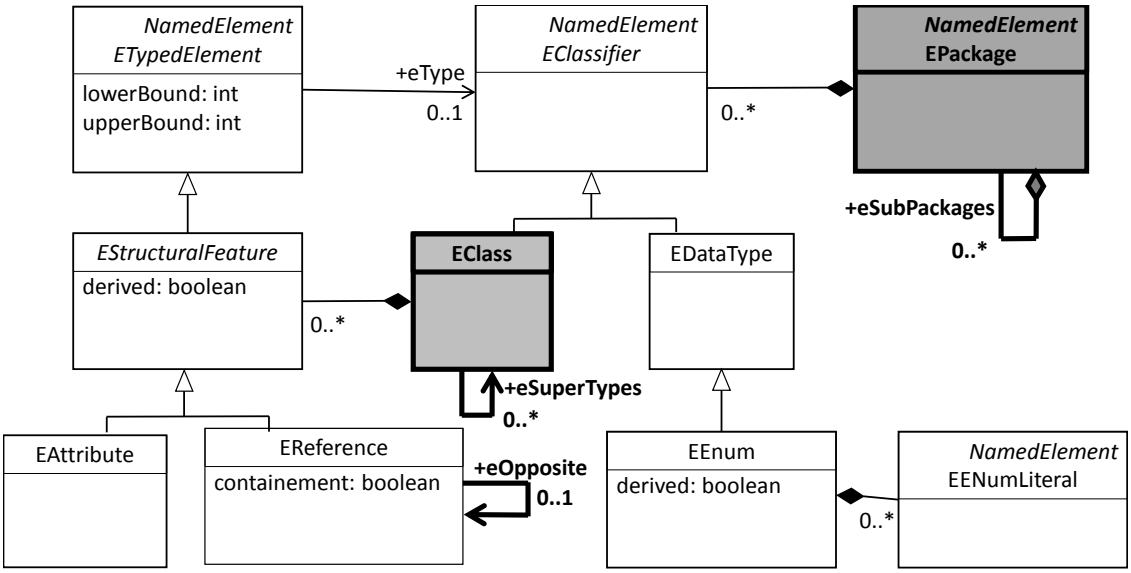


Figure 4.4. Elements of Ecore tagged as mandatory w.r.t. the pattern AcyclicReference

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. 4.5. The same reasoning holds for model refactoring learning or testing. Here the elements to tag are deduced from generic refactoring operations.

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

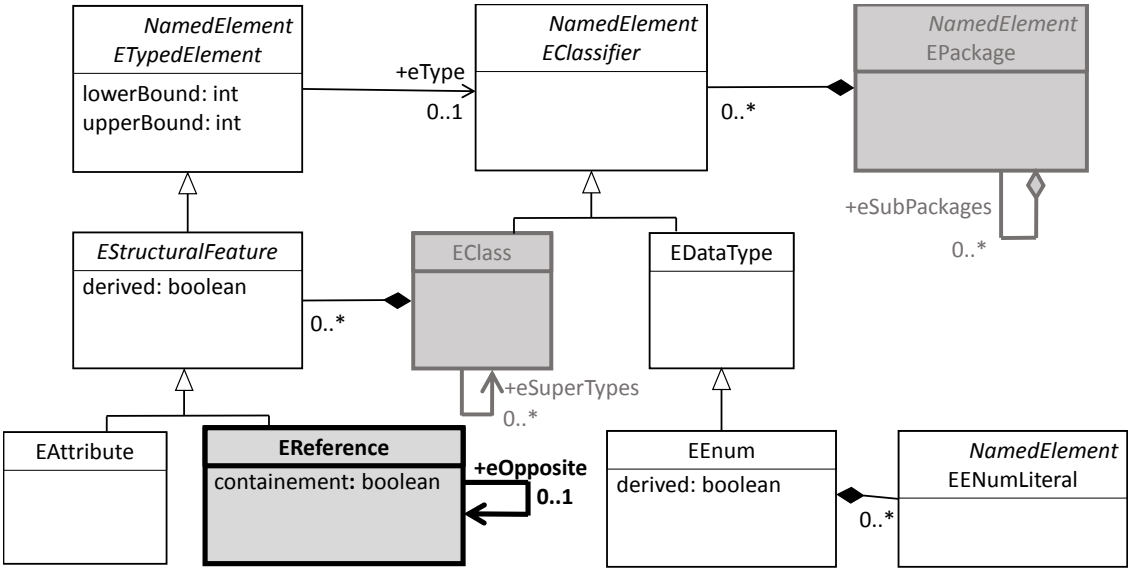


Figure 4.5. Elements of Ecore tagged as mandatory w.r.t. the patterns AcyclicReference and ReferenceDifferentFromSelf

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 (Cadavid et al, 2012), 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 be an advantage, 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

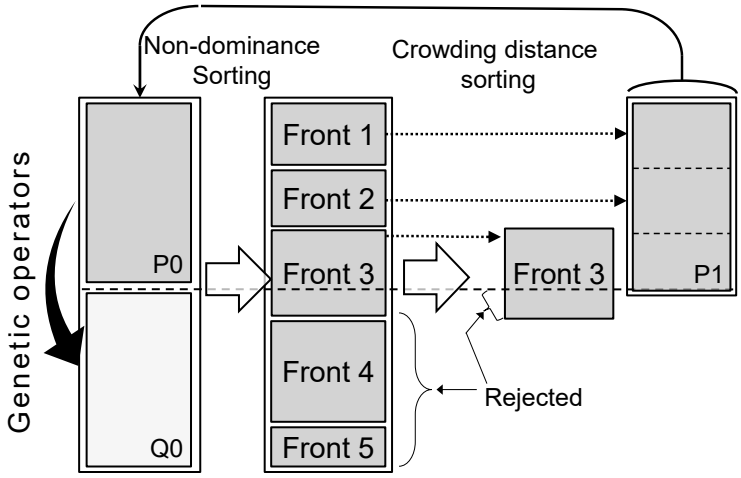


Figure 4.6. NSGA-II (Deb et al, 2000)

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 (Deb et al, 2000). 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* $\forall m \in I = \{1,...,M\}$ $f_m(x) \geq f_m(x^*)$ *and 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 (Deb et al, 2000) 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. 4.6, 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 (Deb et al, 2000). 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 [2]. 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.* (Fleurey et al, 2009). 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.* (Ostrand and Balcer, 1988) 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.

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

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 (Cadavid et al, 2012), formally:

$$DIS = 1 - \frac{excessCovering}{(MFRT \times \#fragmentsCovered)} \tag{3.1}$$

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

$$MFRT = \#fragments \times overlapRatio \tag{3.2}$$

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.* (Cadavid et al, 2012).

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:

$$MIN = 1 - \frac{|ms|}{|MFP|} \tag{3.3}$$

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,

$$MIN_R = 1 - \frac{\sum_{m_i \in ms}(size(m_i))}{|MFP| \times avg\_model\_size} \tag{3.4}$$

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 underestimated, which could results in negative values for MIN and MIN-R. In that situation, 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. 4.7, 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.

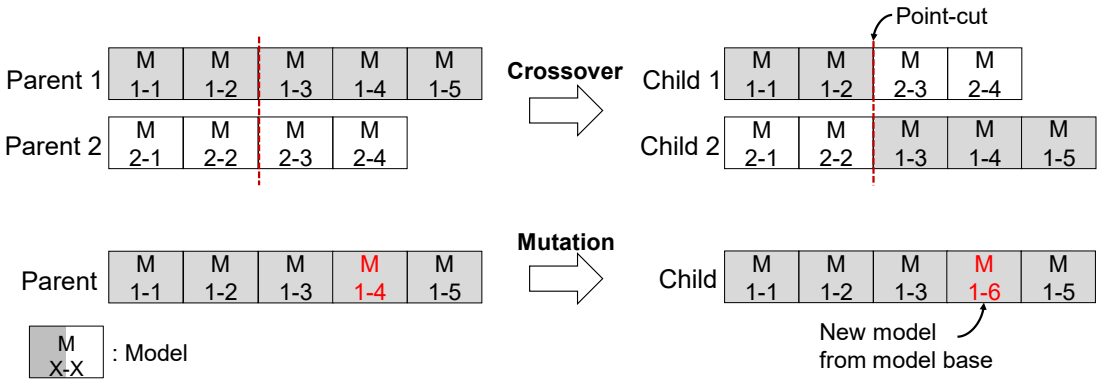


Figure 4.7. Crossover and mutation adapted in our framework

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.* (Wu et al, 2012), 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 (Cuadrado et al, 2012). 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, Feature 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.* (Murashkin et al, 2013). 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

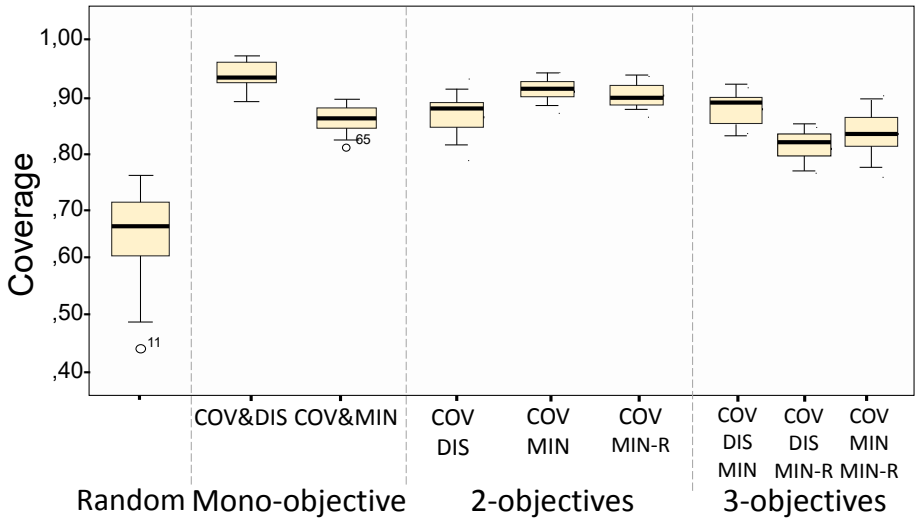


Figure 4.8. Coverage for ATL2.0

Results of the comparison between a random selection and our approach are shown in Figures 4.8 and 4.9. 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. 4.8, 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

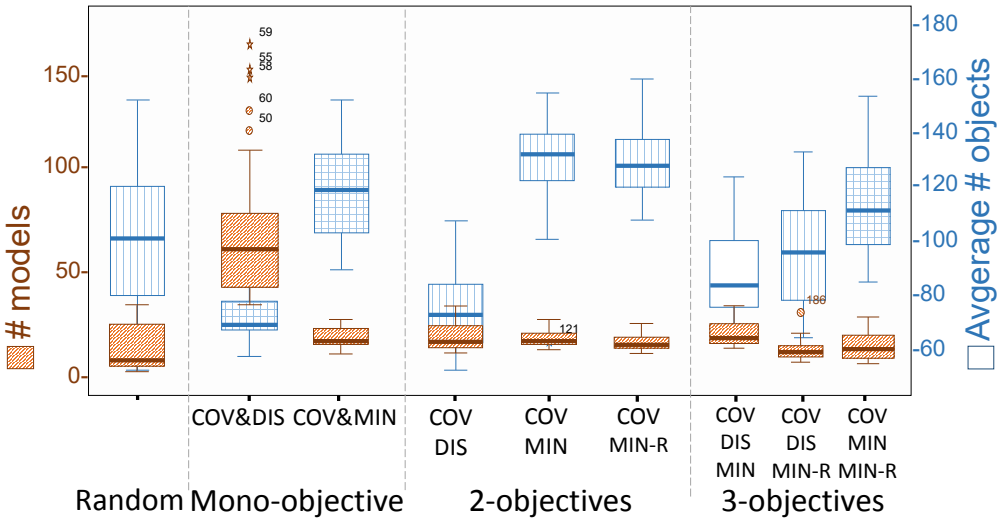


Figure 4.9. Size of solutions for ATL2.0

search and our approach have the best solutions with small model sets (around 18 models per solution) as shown in Fig. 4.9, 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 genetic algorithm approach, we use a single-objective function, defined as the average of the coverage and the minimality values. Formally:

$$f_{mono}(s) = \frac{COV(s) + minimality(s)}{2} \tag{5.1}$$

where $minimality(s)$ can be DIS, or MIN functions as stated in, respectively, equations 3.1 and 3.3. Additionally, we use the crossover and mutation operators of our multi-objective approach with the same probabilities.

In Fig. 4.8 and Fig. 4.9, 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. 4.8 and Fig. 4.9, 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

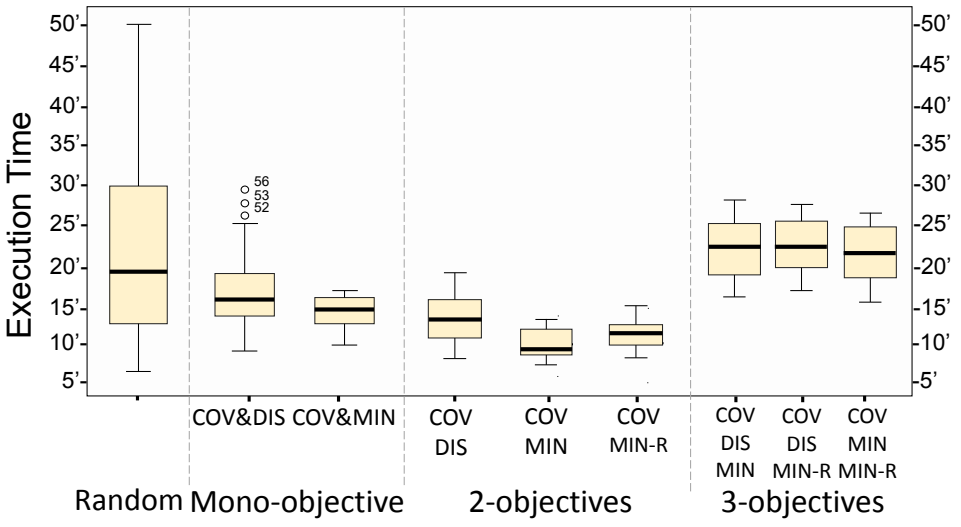


Figure 4.10. Time elapsed during executions

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 larger metamodel (*i.e.,* ATL2.0), the execution time depends on the algorithm and the objectives chosen, as shown in Fig. 4.10.

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 degradation of the coverage. 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 $p\text{-}value < 0.05$. The significance and effect size of the tests used to answer the research questions are summarized in table 4.1. 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

	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: # of models	< 0.01	High	< 0.01	High	0.83	Low
Size: average # of objects	< 0.01	High	0.530	Low	< 0.01	high
Execution time	< 0.01	High	< 0.05	Medium	< 0.01	Low

Table 4.1. 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)

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 respect 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 (Wu et al, 2012), 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.,* (Gonzalez and Cabot, 2014; Gogolla et al, 2015), target specific tasks and purposes without a generalization effort. In particular, Ehrig *et al.* (Ehrig et al, 2006) 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.* (Hoffmann and Minas, 2010). Other teams use constraint logic programming to derive instances (Ferdjoukh et al, 2013; Wu, 2016). 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.* (González Pérez et al, 2012) use a similar approach for the verification of EMF models. Finally, Sen *et al.* (Sen et al, 2009) use *Alloy* (Jackson, 2006) to generate instances satisfying a translated version of the metamodel. However, the UML to Alloy translation is restrictive and challenging as pointed-out in (Anastasakis et al, 2007).
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.* (Selim et al, 2012), 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, (Brottier et al, 2006; Fleurey et al, 2009; Gonzalez and Cabot, 2014)). Gogolla *et al.* (Gogolla et al, 2015) decompose the coverage using classifying terms. Mutation analysis is another way to assess the generated input cases as described (Jia and Harman, 2011; Aranega et al, 2015). Guerra *et al.* (Guerra, 2012) 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.* (Finot et al, 2013)). 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.* (Sadilek and Weißleder, 2008) who consider that the field is not investigated enough. To test metamodels, Cadavid *et al.* (Cadavid et al, 2012) explore the boundaries of the modelling space using an mono-objective evolutionary algorithm. Other teams focus on the interaction between modellers and stakeholders (Cuadrado et al, 2012). Both exhibit the need to have representative, yet small, sets of testing models.

The work of Cadavid *et al.* (Cadavid et al, 2012) falls in the category of search-based testing. Search-based testing aims at defining one or more functions that capture the testing objectives (Anand et al, 2013). 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 (Zhang et al, 2014) 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 (Harman et al, 2015), 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 (Bąk et al, 2013) which finds its root in the idea of a generative perspective on programming (Czarnecki et al, 1997). Model transformation is the most studied field with the early work from Balogh *et al.* (Balogh and Varró, 2009) and Wimmer *et al.* (Wimmer et al, 2007). 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.* (Saada et al, 2012). Similarly, Sun *et al.* (Sun and Gray, 2009) 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.* (Kessentini et al, 2012a) 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.* (Faunes et al, 2013b) and Baki *et al.* (Baki et al, 2014; Baki and Sahraoui, 2016), 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.* (Faunes et al, 2013a). Finally, examples are used for manual modeling and metamodeling activities (Zayan et al, 2014; López-Fernández et al, 2013).

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.

Chapter 5

Injecting Social Diversity in Multi-Objective Genetic Programming: the Case of Model Well-formedness Rule Learning[1]

Edouard Batot and Houari Sahraoui
DIRO, Université de Montréal

[1]The content of this chapter has been published in the 10th Symposium on Search-Based Software Engineering (SSBSE), September 8-9 2018, Montpellier, France. Reference (Batot and Sahraoui, 2018b)

Résumé. Les activités de modélisation logicielle impliquent généralement un effort fastidieux de la part d'un personnel spécialement formé. Ce manque d'automatisation entrave l'adoption du paradigme Model Driven Engineering (MDE). Néanmoins, au cours des dernières années, de nombreux travaux de recherche ont été consacrés à l'apprentissage des artefacts MDE au lieu de les écrire manuellement. Dans ce contexte, la programmation génétique (GP) mono- et multi-objective s'est révélée être une méthode efficace et fiable pour dériver des connaissances d'automatisation en utilisant, comme données d'entrainement, un ensemble d'exemples représentant le comportement attendu d'un artefact. Généralement, la conformité à l'ensemble d'exemples d'entrainement est l'objectif principal pour mener la recherche d'une solution. Pourtant, l'impasse de l'optimum local, l'un des inconvénients majeurs de la GP, perdure lorsque la technique est adaptée au MDE et entrave les résultats de l'apprentissage. Nous cherchons à montrer dans cet article qu'une amélioration de la diversité sociale des populations, réalisée au cours de l'évolution, conduira à une exploration de l'espace de recherche plus efficace, une convergence plus rapide et des résultats plus généralisables. Nous constatons que les améliorations sont dues aux changements appliqués à la stratégie de recherche basée une évaluation empirique dans le cas de l'apprentissage des règles de bonne formation en MDE avec un algorithme génétique multi-objectif. Les résultats obtenus sont frappants et montrent que l'augmentation de la diversité sémantique permet une convergence plus rapide vers des solutions quasi-optimales. De plus, lorsque la diversité sémantique est utilisée pour la distance d'encombrement, cette convergence est uniforme à travers une centaine d'exécutions.

Abstract. Software modelling activities typically involve a tedious and time-consuming effort by specially trained personnel. This lack of automation hampers the adoption of the Model Driven Engineering (MDE) paradigm. Nevertheless, in the recent years, much research work has been dedicated to learn MDE artifacts instead of writing them manually. In this context, mono- and multi-objective Genetic Programming (GP) has proven being an efficient and reliable method to derive automation knowledge by using, as training data, a set of examples representing the expected behavior of an artifact. Generally, the conformance to the training example set is the main objective to lead the search for a solution. Yet, single fitness peak, or local optima deadlock, one of the major drawbacks of GP, remains when adapted to MDE and hinders the results of the learning. We aim at showing in this paper that an improvement in the populations' social diversity carried out during the evolutionary computation will lead to more efficient search, faster convergence, and more generalizable results. We ascertain improvements are due to our changes on the search strategy with an empirical evaluation featuring the case of learning well-formedness rules in MDE with a multi-objective genetic algorithm. The obtained results are striking, and show that semantic diversity allows a rapid convergence toward the near-optimal solutions. Moreover, when the semantic diversity is used for crowding distance, this convergence is uniform through a hundred of runs.

1. Introduction

Model Driven Engineering (MDE) aims at raising the level of abstraction of programming languages. MDE advocates the use of models as first-class artifacts. It combines domain-specific modeling languages to capture specific aspects of the solution, and transformation engines and generators in order to move back and forth between models while ensuring their coherence, or to produce from these models low level artifacts such as source code, documentation, and test suites (Schmidt, 2006). Still, designing and developing artifacts able to perform automated tasks in MDE (ensuring the well-formedness of models, transforming models, etc.) requires one to have both knowledge in the targeted domain as well as in the design and development tools. If done manually, these activities typically involve a tedious and time-consuming effort by specially trained personnel. Such a lack of automation

is considered by many MDE specialists as a threat to MDE adoption (Selic, 2012; Whittle et al, 2014).

Yet, in recent years, many research contributions have shown that it is feasible to automatically learn how to perform a task through examples, or by analogy to similar, previously-solved tasks. More precisely, many of the proposed learning methods are based on Genetic Programming (GP) algorithms, and thereby promise to ease the burden of hand-programming growing volumes of increasingly complex information. As a matter of fact, empirical studies have shown a strong potential in learning automatically model transformations (Kessentini et al, 2011b; Saada et al, 2012; Baki and Sahraoui, 2016) and model well-formedness rules (Faunes et al, 2013a; Batot et al, 2016) from examples of tasks input/outputs. An *example* here must be understood as a couple *<input model; expected output>* defining the constraints that bind artifacts' output to input. The set of training examples represents the expected behavior of the artifact to learn and thus constitutes a convenient objective to lead the search of a solution.

Genetic programming and more generally multi-objective evolutionary computation has received increasing attention in the last decades. From early works (Schaffer, 1985; Goldberg, 1989; Holland, 1992; Koza, 1992), authors have formulated the idea that optimizing for multi-objective is to search for multiple solutions, each of which satisfy the different objectives to different degrees. The selection of the final solution with a particular combination of objectives' values is thus postponed until a time when it is known what combinations exist (de Jong et al, 2001). Studies have shown the value of such techniques and their suitability to real problems. However, from the very beginning, authors pointed out two major drawbacks to the application of genetic programming (GP): (i) diversity of populations is difficult to maintain during evolution, and populations tend to gather around a *single fitness peak*; and, (ii) individuals tend to grow unnecessarily in size – also called *bloating* effect.

Both bloating and single fitness peak symptoms have been well investigated by researchers since early works, and valuable research directions were proposed (Bersano-Begey, 1997; Soule and Foster, 1998; Luke and Panait, 2006). Nevertheless, while adapting GP as an automatic process to learn well-formedness rules from examples, we encountered these same scenarios in a great amount of runs. Solutions agree on finding the correct outputs for a large number of examples, but fail all on a few same examples – a *single fitness peak* is reached.

The approach seems to favor solutions with a high fitness, i.e., a high percentage of correct output found, at the expense of the diversity of the solutions.

On promoting diversity, Vanneshi et al. showed in their work the superior importance of research on indirect semantic methods that *"act on the syntax of the individuals and rely on survival criteria to indirectly promote a semantic behavior"* (Vanneschi et al, 2014). Inasmuch as semantics are considered in GP as a vector of examples, MDE learning from examples methodology offers an auspicious support for such investigations.In the present study, we introduce a new Social Semantic Diversity Measure of individuals (inspired from Natural Language Processing) operating indirectly during the execution of a well-established multi-objective genetic algorithm (Deb et al, 2000). We illustrate our work and assess its value in an empirical study featuring the problem of automatic learning of well-formedness rules from examples and counter examples.

The following section draws a map of the two main drawbacks of genetic programming and how researchers tackle them. Section 3 details how employing our Social Semantic Diversity Measure foster efficiency and accuracy of a GP run. We illustrate our approach in a case study depicted in Section 4. We assess our assumption through an empirical evaluation in Section 5. Section 6 concludes briefly.

2. Background, Related Work, and Problem Statement

Genetic Programming (GP) execution is best understood using Fig. 5.1. At the beginning, an initial population of programs must be created (1). Then, every program of the population is executed on the example inputs, and fitness is evaluated by comparing outputs with the expected ones (2). If a termination criterion is reach (3), the solution program (or a set of near-optimal solutions, in case of multi-objective) is returned (6). Otherwise,

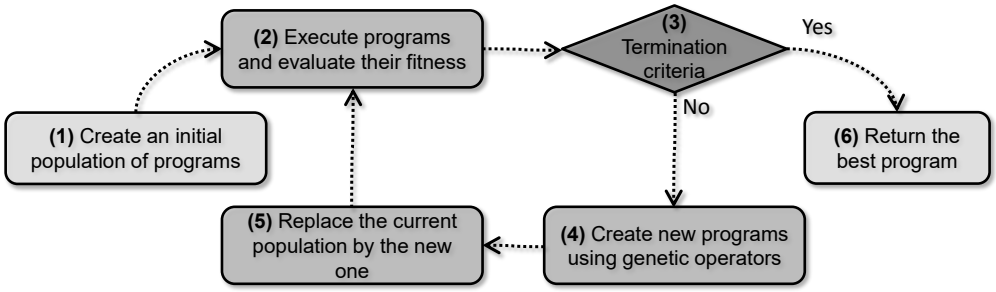


Figure 5.1. A typical genetic programming cycle

a new population of programs is created by genetic operations (crossover and mutation) applied on selected potential reproducers (4). The new population replaces the previous one (5), and a new iteration starts (2). The loop is repeated until a termination criterion is reach (commonly, a perfect fitness, or an arbitrary large number of iterations). Although this process allows to find good solutions for many problems, it is known to suffer from two issues, *bloating* and *single fitness peak*. In the remainder of this section, we briefly discuss the *bloating* issue, and then focus more of the *single fitness peak* issue and its relation to diversity, which is the main object of this paper.

2.1. Bloating

Luke *et al.* suggest that, from a high level perspective, *bloating* (or *code growth*) happens because adding genetic material to individuals is *more* positively correlated to the fitness than removing material. They define it as the "uncontrolled growth of the average size of an individual in the population" (Luke and Panait, 2006). Nonetheless, much work has been done to reduce the effect of bloating, offering to present readers a few options to choose from (Soule and Foster, 1998). More precisely, in a multi-objective context, Pareto-based Multi-objective Parsimony Pressure (*i.e.,* using an objective devoted to constraining size of individuals) has been found very effective – with limited side effects (de Jong et al, 2001; Ekárt and Németh, 2000). We use this technique in our experiments.

2.2. Single fitness peak

The second problem with GP is the risk of a *single fitness peak* (de Jong et al, 2001), consisting in a premature convergence together with a loss of diversity. Candidate solutions get stuck in a local optima and often no further improvement in fitness is noticed (Wyns et al, 2006). To tackle this issue, the level of diversity a population conveys must be given due consideration during a GP run (Burke et al, 2004). More precisely, two phases of such a run are appropriate: at the initial population creation, to ensure a broad genetic material base; and/or during the evolution itself, to ensure that diversity does not fall from one generation to the next. In both cases, diversity exists in two kinds: genotypic diversity considers the level of variability in individuals' structure, whereas phenotypic diversity focuses on the behavior of individuals.

2.2.1. *Genotypic Diversity*

Genotypic diversity is the variety of individuals among a population with regards to their structure. It's a measure of the distance between individuals' syntax (McPhee and Hopper, 1999; McPhee et al, 2008). MDE though, since the syntax of artifacts is (very) complex, does not bare a single consensual definition of genotypic (or structural) diversity (Baudry and Monperrus, 2015; Giraldo et al, 2014). Nonetheless, to bestow a sufficiently diverse genetic material to start an evolutionary computation with, teams have used different metrics based on coverage estimations and showed interesting results. Works vary in nature and offer automatic generation of *diverse* models (Batot et al, 2016; Wu, 2016), or a user visual assistance helping when eliciting learning inputs data (Ferdjoukh et al, 2017; Cuadrado et al, 2012; López-Fernández et al, 2015). In any case, both techniques can be employed to provide with diverse initial population of solutions as well as with qualified input data.

2.2.2. *Phenotypic Diversity*

As opposed to genotypic diversity, phenotypic diversity is measured on the behavior of a program – independently to its syntax. A phenotypic (or semantic (Vanneschi et al, 2014)) measure, refers to the proportion of examples correctly processed by a program (*i.e.,* producing the expected output when executed on a specific input). It is a tangible fact that phenotypic diversity is more efficient than genotypic diversity to avoid the single fitness peak problem (Vanneschi et al, 2014). Nonetheless, if some early studies went as far as to expand the Darwinian metaphor and considered preference between individuals during GP run (Ryan, 1994), to the best of our knowledge, there exists no study explicitly measuring benefits of phenotypic diversity when learning MDE artifacts.

2.2.3. *Indirect Semantic Diversity Methods*

Roughly speaking, these methods combine both genotypic and phenotypic diversities. The rationale behind indirect diversity methods lies in their ability to distinguish between the aim of the method: individuals with acute Semantic Fitness, and the mean of its application: genetic modifications performed on their syntax. Understood as such, the heuristic remains agnostic of its mean of achievement and is ready to convey a strong generalization potential (Dabhi and Chaudhary, 2012). Vanneshi *et al.* (Vanneschi et al, 2014) have proven

the power of indirect diversity methods and call for more research in this field. It is to note here that, in the context of learning artifacts from examples in MDE, Semantic Fitness measure is a built-in feature and comes at no extra cost.

3. Social Semantic Diversity Measure

Notwithstanding that MDE-artifact learning from examples might be perfectly fit to GP adaptation, single fitness peaks yet keep happening during evolution. This leads to a disproportionate number of solutions with a good fitness, at the expense of their diversity. Processing most examples correctly, these *alphas* (Bersano-Begey, 1997) struggle to solve all examples exhaustively. Meanwhile, unfortunately, solutions able to solve the remaining corner cases reach a (much) lower fitness. Withal, since reproducers are chosen with regard to their fitness, the genetic material these latter *partial solutions* convey is lost and *corner cases* are never solved. A remedy to this deficiency was found using a social diversity measure.

We call *Social Diversity Measure* a measure that does not take into account the only individualistic fitness (*i.e.,* how many examples an individual resolves) but considers as well a social dimension (*i.e.,* what does that individual bring to the general fitness of the population).

Since we use a Semantic fitness, the remaining of this paper will mention Social Semantic Diversity Measure (SSDM). Its computation, based on the *inverse example resolution frequency* (IERF) is inspired from the *term frequency-inverse document frequency* (*TF-IDF*) numerical statistic (Sparck Jones, 1988) from information retrieval research field. In other words, the SSDM of a solution is the sum of IERF of the examples it solves.

Paraphrasing *TF-IDF* definition may help the reader to grasp the general idea of SSDM. We formulate it as follows: "SSDM increases proportionally to the number of examples solved and is offset by the frequency of which an example is solved by the population's individuals, which helps to adjust for the fact that some examples are more frequently solved in general."

As a consequence, SSDM favors solutions solving *corner cases* by considering how many solutions in the population solve an example.

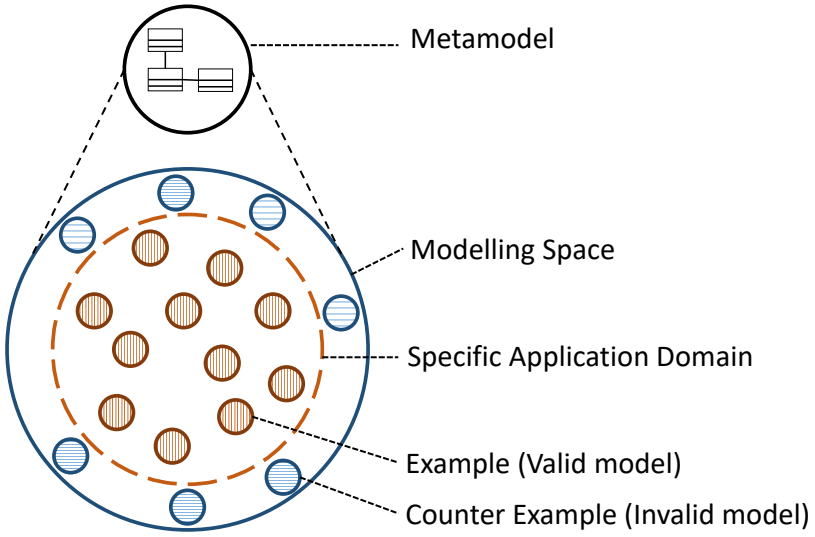


Figure 5.2. Metamodel, modelling space and application domain

4. Learning Well-formedness Rule

In this section, we illustrate how social semantic diversity can be implemented in a multi-objective genetic-programming algorithm to learn well-formedness rules (WFRs) from examples. As mentioned in the introduction, researchers offer to use GP to learn some of MDE artifacts automatically as a substantial alternative to writing them manually. Indeed, we aim at showing in this paper that, during the process, which scalability remains at stake (Harman et al, 2015), an improvement in populations' social diversity will lead to more efficient search and more generalizable results. Thus far, the reader is asked to understand the little space left for implementation details.

After a brief overlook at the use and function of well-formedness rules, we will depict how much a tangible support GP, and more precisely multi-objective GP, offers to learn them automatically from examples and counter examples.

4.1. Well-formedness rules

In the MDE paradigm, due to their high level of abstraction, metamodels usually define too-large modelling spaces. They must be enriched with constraints, or rules, limiting the scope of their possible instantiations, *i.e.,* well-formed models in contrast to ill-formed models. Fig. 5.2 schematizes the concept of specific application domain: a metamodel defines a modelling space (within blue line) ; of which a specific application domain is a sub-space (within red dashed line). A set of WFRs allows to automatically differentiate between valid

(well-formed) and invalid (ill-formed) models – it formally describes the limit of that targeted specific domain.

Representation. In the context of a GP learning process, a solution to our problem is thus a set of WFRs. More precisely, we represent a WFR as a tree which nodes are logical operators (*AND*, *OR*, *IMPLIES*, and *NOT*) and first-order quantifiers (*forAll* and *exists*), and which leaves are learning atomic blocks in the form of *OCL patterns* instances. Consequently, a solution is a tree with as root a vector whose elements are pointers to the individual WFR trees. Fig. 5.3 shows an example of a candidate (not necessarily valid) solution with 3 WFRs for the state-machine metamodel. The first and second rules constrain a *final* state to have respectively one incoming transition and no outgoing transition. The third rule requires that a pseudostate *choice* must have at least one incoming or outgoing transition. As for their execution, we implement WFRs in the *defacto* language Object Constraint Language (OCL[2]).

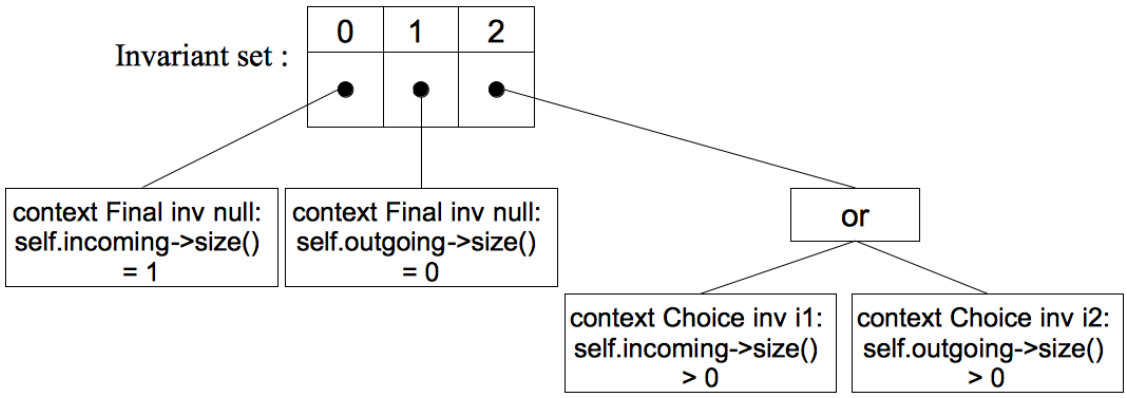


Figure 5.3. An example of solution containing 3 WFRs.

OCL patterns. The rationale behind OCL patterns is beyond the scope of this paper. They result from empirical studies carried out on more than 400 metamodels from industry and academe alike (Cadavid et al, 2015). In a nutshell, OCL patterns should be understood here as a minimalistic set of templates which instantiation and composition allows to express all and every *useful* WFR.

Size concern. Since solutions must be legible by final user (*i.e.,* within human reach), the size and number of constraints must be kept as small as possible.

[2]http://www.omg.org/spec/OCL/

4.2. GP Adaptation

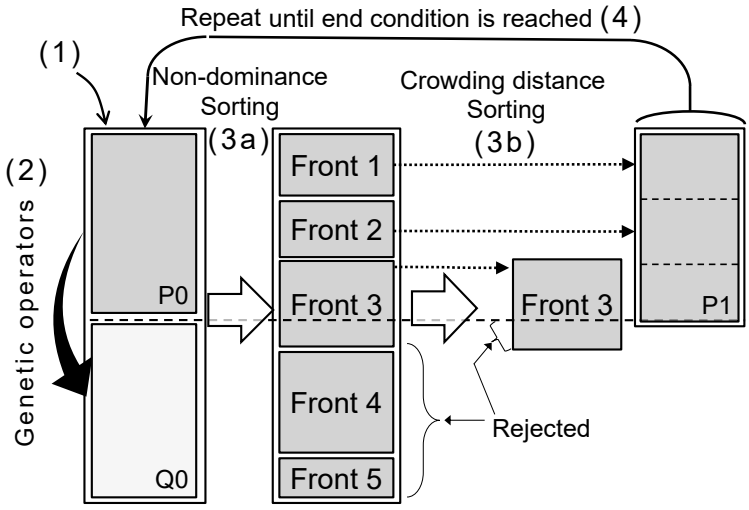


Figure 5.4. Non Sorting Genetic Algorithm NSGA-II (Deb et al, 2000)

Our goal is to find the minimal set (*i.e.,* size) of WFRs that best discriminates between the valid and invalid example models (*i.e.,* fitness). Size and fitness objectives being contradictive in nature, we represent the learning of WFRs as a multi-objective optimization problem, and we solve it using the Non-Sorting Genetic Algorithm NSGA-II (Deb et al, 2000).

The idea of NSGA-II (Deb et al, 2000) 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. 5.4, the first step in NSGA-II is to create randomly a population P_0 of $N/2$ individuals encoded using a specific representation (1). 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 (2). Both populations are merged into an initial population R_0 of size N, which is sorted into dominance fronts according to the dominance principle (3a). A solution s_1 dominates a solution s_2 for a set of objectives $\{O_i\}$ if $\forall i, O_i(s_1) \geqslant O_i(s_2)$ and $\exists j \mid O_j(s_1) > O_j(s_2)$. 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 (3b). This process will be repeated until a stop criterion is reached, *e.g.,* a number of iterations or a certain value of the Semantic Fitness.

We adapted NSGA-II to our problem as follows.

- **Solution Representation and Creation.** A solution to our problem is represented as mentioned in Section 4.1, *i.e.,* a set of OCL constraints, each implementing a WFR represented as a tree. The initial population is created randomly. For each individual, the average number of nodes in the WFR trees, the maximum depth, and the maximum width are configurable;
- **Reproduction.** As genetic operators, we use a single-point crossover applied to the tree-root vector, and two kinds of mutations. First, a node from a WFR tree is chosen randomly. If it is a leaf, the pattern instance is either replaced with a new randomly created one or, if applicable, the pattern parameters are replaced randomly with applicable values. If the selected node is a logical operator, this is changed randomly.
- **Objectives.** We consider three objectives: *Size* is the number of leaves in the constraint tree, the smaller the better; *Semantic Fitness* is the number of examples processed accurately by an individual, to be maximized; and *Diversity* is SSDM, which can be represented either as an objective or a crowding distance, to be maximized as well.
- **Termination criteria.** Evolution stops if either a Semantic Fitness of 99%, or an arbitrary large number of iterations, is reach.

4.3. Social Semantic Diversity Implementation

We offer to employ the Social Semantic Diversity Measure (SSDM) in two different ways. The first is as an objective of its own, considered together with above-mentioned size and fitness (as promoted by Dejong *et al.* (de Jong et al, 2001)). The other builds on peculiar limitation of NSGA-II (Fortin and Parizeau, 2013) and acts as an alternative to the computation of a crowding distance. In both cases, SSDM computation remains the same.

More specifically, implementing SSDM comes to adapting TF-IDF (Sparck Jones, 1988) using examples as documents and solutions as words. This is detailed in Listing 5.1. At a given iteration, SSDM is calculated from a binary matrix in which each cell represents the score of an individual against an example of the training set. The frequency of an example is the number of times it is solved by individuals (first *for* loop). Finally, individual's SSDM value is the sum of *inverse example resolution frequencies* of examples that it processes accurately (last *for* loop). More precisely, variables are:

- $example\_set$, the vector of training examples;
- $sol\_vs\_examples$, which contains the result of the comparison between output of individuals and output of the oracle when executed on $example\_set$;
- and fq_ex, which contains examples frequencies, recording how many solutions solve each example from $example\_set$;
- $ierfi$, the vector of *inverse example resolution frequencies* of training examples.

5. Evaluation

To assess the improvement brought by our social semantic diversity in the search strategy, we conducted an empirical evaluation[3]. We formulate our research questions as follows:

- **RQ0**: Are our results a consequence of an efficient exploration of the search space, or are they due to the vast number of individuals we consider during the evolution?
- **RQ1**: Does the use of Social Semantic Diversity as an objective improves the search strategy, and, if so, how much?
- **RQ2**: Does the use of Social Semantic Diversity as an alternative crowding distance exhibit better efficiency and generalizabilty than as an objective?

5.1. Setting

In order to mitigate the influence of a metamodel specific structure on the learning process, we selected three metamodels (`FamilyTree`, `Statemachine`, and `ProjectManager`) that demonstrate different levels of structure complexity and require diverse OCL WFR sets. We provided with oracle (*i.e.,* expected WFRs) manually. In more details, `FamilyTree` is the most simple case. Yet, it has been used as an illustrative example in various publications

[3]All experiment data is available at http://www-ens.iro.umontreal.ca/~batotedo/ssdm_exp/

Listing 5.1. Excerpt for SSDM weights calculation.

```
\\ Compute frequencies of examples solved
for (int i = 0; i < sol_vs_ex.length; i++)
for (int j = 0; j < sol_vs_ex[i].length; j++)
fq_ex[j] += sol_vs_ex[i][j];

\\ Inverse document frequencies
for (int j = 0; j < fq.length; j++)
ierfi[j] = Math.log10(D/fq_ex[j]);

\\ Weigthing
weight = 0;
for(int j = 0; j < example_set.length; j++)
if(example_set[j].isAccurate())
weight += ierfi[j];
```

in the MDE research literature, such as (Gogolla et al, 2015). `Statemachine` illustrates structural cardinality restrictions and define a common, widely used language. Finally, `ProjectManager` is the most complex case and comes from (Hassam et al, 2010).

5.1.1. *Learning examples*

To provide with example sets of *quality* (*i.e.,* covering at best the modelling space, yet as small as can be), we used a model generator (Batot et al, 2016). Size matters since every generated model example must be, in a real setting, tagged manually as *valid* or *invalid*. For the sake of experiment, we use the WFRs oracles to mimic the manual tagging. To run the experiment, we used two sets of examples for each metamodel. On the one hand, 20 models (10 valid, 10 invalid) were required for the learning (a *training set*). On the other hand, a *test bench* of 100 models (50 valid, 50 invalid) was used to measure solutions' accuracy (or generalizability).

5.1.2. *Configurations and variables*

Four configurations were considered to illustrate and answer our research questions (see Section 4.2 for implementation details). **RND** is a random exploration of the search space that takes the best among a given number of solutions randomly generated; **STD** is a standard run of NSGA-II (Deb et al, 2000) with two objectives, size and semantic fitness; **OBJ** is a run of NSGA-II with three objectives: size, semantic fitness, and SSDM diversity;

and **CD** is a run of NSGA-II with size and semantic fitness as objectives, and SSDM as crowding distance.

We used two dependent variables to quantify experiment results: **#GEN**, the number of generation the evolutionary computation needed to find a solution. A score of 3000 means that there was no solution with perfect fit found during the search, and **ACC**, the proportion of examples from the test bench a solution process accurately.

5.1.3. *Evaluation protocol*

For the NSGA-II parameters, we use a maximum number of iterations of 3000 and a population size of 30 solutions. Crossover and mutation probabilities are set to 0,9 and 0,3 respectively. In addition, solutions are created with between 5 to 15 WFRs with each WFR having a maximum depth of 3 and width of 15. We answer RQ0 with a comparison between the results given when using SSDM as an objective (OBJ) in the search strategy and those of a random exploration (RND). Since our strategy explores 3000*30 solutions, the random exploration explores randomly 90000 solutions as well and considers the best individual so created. We answer RQ1 with a comparison between the solutions obtained after an execution with and one without social semantic diversity objective (respectively OBJ and STD). Finally, we answer RQ2 by comparing the configurations with social semantic diversity objective (OBJ) and with social semantic diversity crowding distance (CD). We ran each treatment 100 times to tackle GP indeterminism and we guarantee statistical significance of the findings using the Mann-Whitney test.

5.2. Results and Analysis

5.2.1. *RQ0 - Sanity Check*

As can be seen in Table 6.1, the RND configuration gives very poor results in comparison with an OBJ execution for the two most complex metamodels (average accuracy on test bench is 0.5 vs 0.76 for `ProjectManager` and 0.53 vs. 0.94 for `Statemachine`). The difference in both cases is statistically significant (*p-value*<0,001) and the effect size is large (*Cohen's d* > 5). For the small metamodel `FamilyTree`, although statistically significant, the difference and the effect size are small. ***We can conclude that solutions are significantly more generalizable when using OBJ configuration***.

Table 5.1. Statistical comparison of results between random search and our approach on three WFR learning scenarios.

	Average ACC Value		Mann Witney p-value	Effect Size Cohen's d
	RDN	OBJ		
`ProjectManager`	0.5	0.76	<0.001	7.35
`Statemachine`	0.53	0.94	<0.001	5.38
`FamilyTree`	0.93	0.98	<0.001	0.74

5.2.2. *RQ1 - Social Semantic Diversity Method, an improvement?*

Efficiency shows a significant improvement when SSDM is used, as can be seen in odd columns of Fig. 5.5. The number of generations required to find a solution when employing OBJ is a lot smaller than when employing STD. With `ProjectManager` metamodel, an STD run hardly find solutions solving all training examples within 3000 generations, but OBJ do it in an average of 260 generations. More, solutions were found with significantly better accuracy than STD (respectively 0.76 against 0,69) and thus strengthen solutions' generalizability likewise. This success is also noticed, if of lesser magnitude, during executions on the `Statemachine` metamodel. Here, if solutions are found in both configuration, yet OBJ is significantly faster (with 782 generations, when STD requires more than 1782). As for the `FamilyTree` metamodel (not shown if the figure), solutions given by OBJ executions output a similar ACC (0.98) but significantly faster with 25 generations (resp. 76 with STD). ***We can conclude that injecting the social semantic diversity significantly improves the learning results***.

5.2.3. *RQ2 - Social Semantic Diversity Method as an alternative crowding distance, any better yet?*

Results of RQ2 are flagrant (see the third configuration for both metamodels in Fig. 5.5). A hundred runs show together how using SSDM (Figures 5.8 and 5.7) surges the learning curves and fosters solution exploration compared to a standard run (Figure 5.6). As for generalizability, it doesn't seem that choosing between SSDM as an objective (OBJ) or in the crowding distance (CD) has any significant impact on the accuracy of solutions on test bench found (Mann Witney p-value > 0.01; see even columns in Fig. 5.5 for an illustration). Thence,

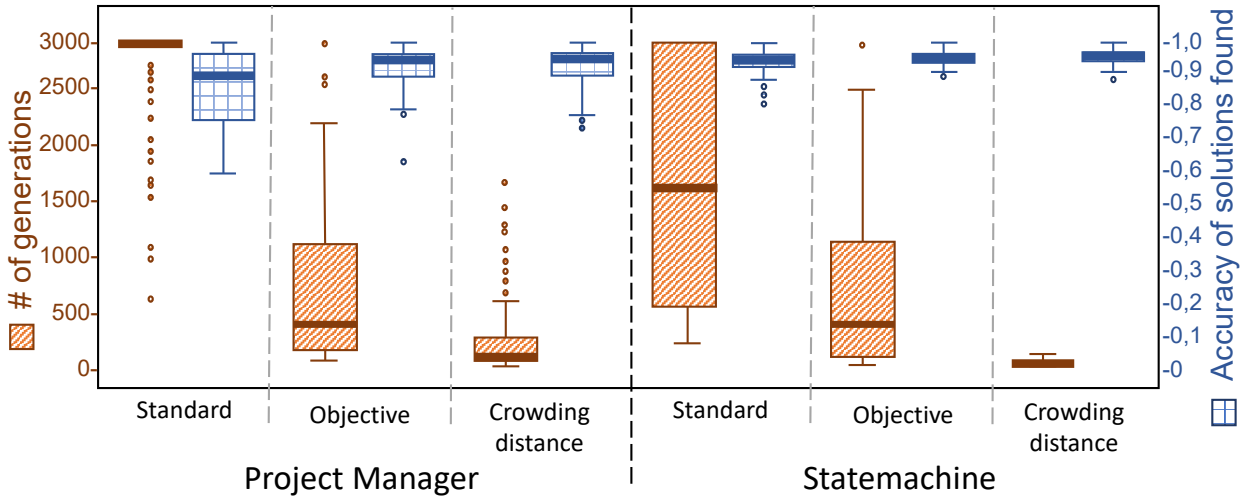
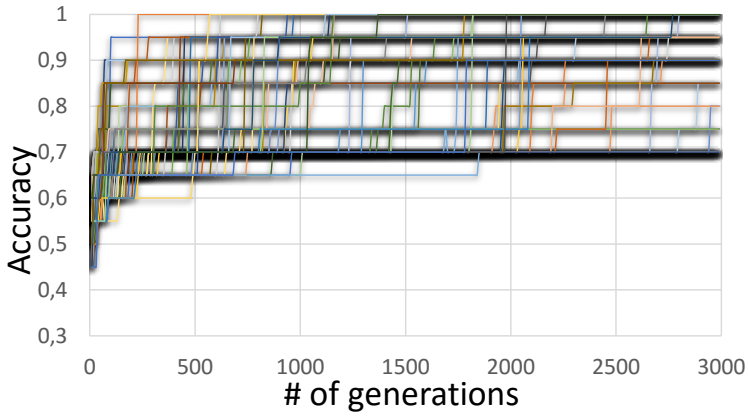


Figure 5.5. Number of generations to find solutions and their accuracy on test bench for `ProjectManager` and `Statemachine` metamodels.

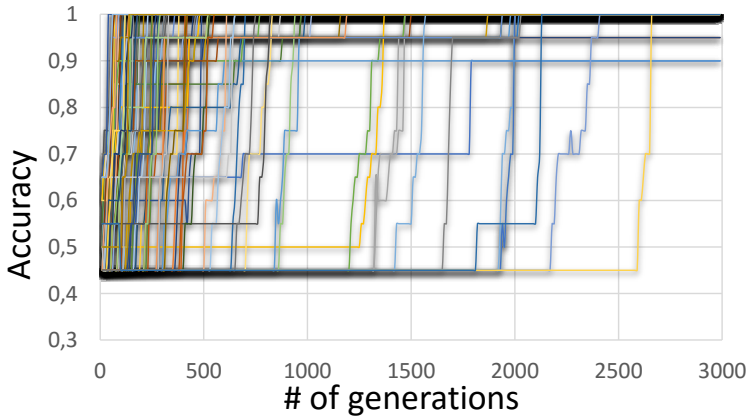


5in

Figure 5.6. Evolution of individuals' average accuracy value during runs on `ProjectManager` metamodel, with a hundred runs a plot. Standard evolution (STD)

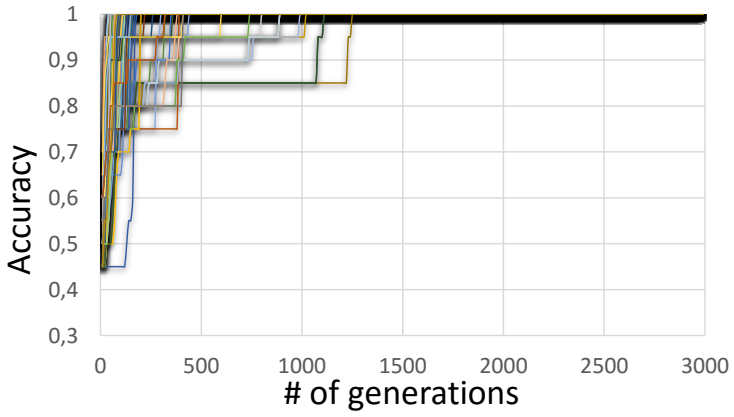
the main difference lies in the smaller average number of iterations CD needs to converge, compared to OBJ runs. Note that that analysis is the strongest with `ProjectManager` and `FamilyTree` metamodels. With `Statemachine` metamodel's results are slightly mitigated but remains significant. In that case, WFRs are more generally focused on structural cardinality than WFRs of the two other metamodels. We conceive this might be a factor for slightly different results. ***We can conclude that social semantic diversity as a crowding distance is more efficient than as an objective***.

In conclusion, as shown in Fig. 5.5 and Figures 5.6, 5.7, and 5.8, and certified with statistical analysis, the OBJ strategy surpasses significantly a STD exploration of solutions. Convergence is faster and output more generalizable (*i.e.,* confronting solutions to a test



5in

Figure 5.7. Evolution of individuals' average accuracy value during runs on `ProjectManager` metamodel, with a hundred runs a plot. SSDM as an objective (OBJ)



5in

Figure 5.8. Evolution of individuals' average accuracy value during runs on `ProjectManager` metamodel, with a hundred runs a plot. SSDM as crowding distance (CD)

bench gives better results). A reason for these results might come from the way size is controlled. As recognized in the literature, we implemented it as a Pareto-based Multi-objective Parsimony Pressure. We noted, as expected (de Jong et al, 2001), that solutions were skewed toward a 1.0 size, and the Pareto front grew large. Solutions' size was indeed the one expected (*i.e.,* legible by a human), and the search, passed a few generations, relied mainly on Semantic Fitness. As a presumed consequence, when putting SSDM as an alternative to crowding distance, results were breathtaking on the three metamodels. Finally, using Social Semantic Diversity Measure as an alternative crowding distance outperforms its use as an additional objective. Convergence is boosted, and generalizability is kept at its maximum. We hope these results are generalizable and claim the need to explore other applications, with OBJ and CD alike.

5.3. Threats to Validity

Although our approach produced good results on three metamodels, a threat to validity resides in the generalization of our approach to other scenarios. Still, metamodels show different characteristics and origins, and while our sample does not cover all learning scenarios, we believe that it is representative enough of a wide range of metamodels.

Another threat to the validity of our results relates to the use of a single set of (20) models to learn each WFR sets. Characterization of example sets is an ongoing investigation, and different sets might show different results. Yet, to mitigate what specificities the manual design of models can bring and encourage replication of our work, we used a generator (Batot et al, 2016). Also, using the same set in every configuration ensures a difference in sets do not interfere in the experiment.

Regarding the applicability to other MDE artifacts, we believe that the idea to consider the social dimension of individuals' characteristics shall apply to the evolutionary computation of model transformation as well. In this case, *inverse example resolution frequency* could be used as well and we prospect, as future work, to replicate this study on model transformation learning.

6. Conclusion

This paper studies the impact of using a social semantic diversity to improve the search process for the multi-objective optimization problem of learning model well-formedness rules from examples and counter examples. The *Social Semantic Diversity* is measured (SSDM) in a way that does not take into account the only individualistic fitness (*i.e.,* how many examples an individual resolves) but considers as well a social dimension (*i.e.,* what does that individual bring to the general fitness of the population). We integrated SSDM in the NSGA-II algorithm as (i) an additional objective, and (ii) as an alternative to the crowding distance.

We evaluated the two options by learning WFRs for three metamodels. Our results are compiling evidence that injecting the social semantic diversity in the search process, especial as an alternative to the crowding distance, improves the convergence and the quality of the learned artifacts. The proposed measure and its integration in the multi-objective optimization algorithm are unaware of the learned artifact and the input/output examples

used to guide the search. This allows to use social semantic diversity for a wide range of problem that can be solved by a multi-objective genetic programming algorithm. This claim must, however, be supported by replication studies. We expect to conduct some of these studies, especially for model transformation learning. Finally, we encourage further replication of our work to determine whether different multi-objective GP algorithms could benefit as well from our discovery.

Chapter 6

Characterizing Example Sets for Knowledge Derivation in Model-Driven Engineering[1]

Edouard Batot and Houari Sahraoui
DIRO, Université de Montréal

[1]The content of this chapter has been submitted to the ACM Transactions on Software Engineering and Methodology journal (TOSEM). Reference (Batot and Sahraoui, 2018a)

Résumé. Dans le but d'accroître le niveau d'abstraction des langages logiciels, l'Ingénierie Dirigée par les Modèles (IDM) se base sur l'automatisation des tâches de manipulation et d'opérationnalisation des modèles, telles que la génération de code, les transformations et le contrôle de conformité, est indispensable pour permettre une utilisation concrète des modèles. En effet, un outillage concret capable d'automatiser ces tâches est indispensable pour permettre une utilisation concrète des modèles. Malheureusement, les algorithmes et les artefacts capables d'automatiser ces tâches deviennent rapidement très complexes et leur édification nécessite des connaissance de différentes natures - des compétences en modélisation et une maîtrise du domaine spécifique - qu'il est difficile, voire impossible, de trouver en la même personne.

Au cours des dernières années, des équipes de chercheurs se sont efforcées de résoudre ce problème en utilisant des jeux dexemples dapplication pour apprendre automatiquement des connaissances abstraites telles que la transformation de modèle, la refactorisation, la localisation des défauts et les règles de bonne formation. Malgré les résultats encourageants, ces équipes utilisent des exemples soit (i) tirés de cas industriels, soit (ii) construits spécifiquement pour la validation de leur approche. La représentativité des exemples par rapport à l'espace des problèmes n'est pas systématiquement évaluée et la validité des résultats est donc incertaine. En parrallèle, les tests de transformation de modèles indiquent que la *couverture* est un candidat de premier choix pour la mesure de la représentativité des jeux d'exemples.

Dans cet article, nous proposons une méthodologie complète pour apprendre les artefacts IDM à partir dexemples et nous étudions la relation entre la représentativité des exemples en termes de couverture de lespace du problème et la qualité de solutions en termes dexactitude et de généralisabilité. Nous évaluons empiriquement cette relation dans une étude de cas: l'apprentissage automatique à partir d'exemples et de contre-exemples de règles de bonne formation. Les résultats montrent à quel point la généralisabilité des solutions est subordonnée à la couverture des exemples. Modérés mais significatifs, ces résultats préliminaires laissent présager dautres travaux intéressants sur la taille et la couverture, ainsi que sur la diversité des artefacts de l'IDM.

Abstract. With the aim of raising the level of abstraction of software languages, Model Driven Engineering (MDE) paradigm promotes automation as one of its founding principles. Indeed, a concrete tooling able to automate model manipulation and operationalization tasks such as code generation, transformations, and conformance check is mandatory to allow an efficient use of models. Unfortunately, algorithms and artifacts able to automate these tasks become quickly very complex and their edification requires knowledge of different natures – from modelling skills to specific domain expertise – that is difficult, if not impossible, to find within the same person.

In recent years, teams of researchers have been striving to address this issue using sets of application examples to automatically learn abstract knowledge such as model transformation, refactoring, defect localization, and well-formedness rules. Notwithstanding the encouraging results, these teams use examples either (i) picked from industrial cases, or (ii) built specifically for their approach. Yet, the used-examples representativeness of the problem space is not assessed systematically.

In this paper, we propose a complete methodology to learn MDE artifacts from examples and we investigate the relationship between the representativeness of examples in terms of their coverage of the problem space and the quality of solutions in terms of accuracy and generalizability. We empirically evaluate this relationship in a case study: the automatic learning of well-formed rules from examples and counter-examples. Results show how subordinate the generalizability of solutions is to the coverage of examples. Moderate yet significant, these preliminary results augur further interesting work on the size and coverage, as well as on diversity of MDE artefacts.

1. Introduction

Model Driven Engineering (MDE) aims at raising the level of abstraction of software languages. It promotes the use of modelling languages tailored to domain-specific needs. Fundamentally, a Domain-Specific Modelling Language (DSML) forms a knowledge definition which is legible by domain experts. Over the last decade, MDE has proven its potential as it enhances communication, alleviates development complexity and productivity (Hutchinson et al, 2011a,b).

Automation is a founding principle of MDE since DSMLs are intended to be used with concrete tools that automate model manipulation and operationalization tasks such as code generation, transformations, and conformance check. Unfortunately, artifacts able to automate these tasks become quickly very complex and their edification requires knowledge of different natures. Indeed, in addition to have a solid expertise in the application domain, the developer must be knowledgeable in modelling and must be familiar with the artifacts' languages. This inherent complexity remains a "significant entry barrier to MDE adoption", as mentioned by Paige *et al.* (Paige et al, 2017). Writing manually each artifact has proven being too risky as well since a high level of complexity makes artifacts error prone. Additionally, unlike for general modeling languages such as UML, the critical mass of knowledge required for automation cannot be found for every single specific domain specific language.

An alternative to manually write automated artifacts is to learn them from input/output examples. Example-Driven Modelling (Bąk et al, 2013) shows that knowledge necessary to automation can be derived from application examples. These examples are formed of models representing the possible inputs of a model operationalization/manipulation task and the corresponding expected output. For instance, an example of a model transformation from a DSML to another language is an instance of the corresponding source metamodel coupled with its expected transformed instance in the target metamodel.

In recent years, many research teams have shown the feasibility of using sets of examples to concretely learn abstract knowledge such as model transformation, refactoring, defect localization, and well-formedness rules. They use examples picked from industrial cases or built specifically for this purpose. Yet, since the relation between example sets and solution quality is obvious, which characteristics or properties of the examples are of consequence?

The objective of this paper is then to study the representativeness of application examples in order to improve the quality of MDE artifacts learning. We evaluate the relation between the characteristics of examples in terms of representativeness of the problem space and the quality of solutions in terms of accuracy and generalizability. To this end, we use the complex learning problem of well-formedness rules from model examples and counter-examples.

More specifically, our work takes the form of three contributions:

(1) Characterization of learning MDE artefacts by examples problem;

(2) Improved algorithm for learning automatically well-formedness rules (WFR) from examples and counter examples;

(3) Empirical evaluation of the relation between representativeness of examples as it is considered in literature, and accuracy/generalizability of solutions in the case of WFR learning on three different cases.

The remainder of the paper is organized as follows. After introducing the thorough learning process in Section 2, we illustrate its main characteristics through the case of learning Wellformedness rules from examples and counter examples in Section 3. Section 4 shows the results and analysis of the empirical evaluation of the relation between examples representativeness and the inductive power of the resulting learned artifacts. Related work is presented in Section 5 before we conclude in Section 6.

2. Learning Complex Artifacts

In this section we will detail a process to automatically learn MDE tasks by use of examples. First, we will present the thorough learning process before detailing the nature of examples themselves. We then addressthe possibilities of automation of the process and their consequences. Finally, we will develop a strategy to reduce the research space to apprehensible dimensions.

2.1. Learning Principle

To help automate the tasks involved in model manipulation in the MDE paradigm, research teams have shown feasible to use Search Based techniques to learn these tasks automatically from examples. Tasks learned are performed by artifacts such as model transformations, wellformedness rules, or refactorings.

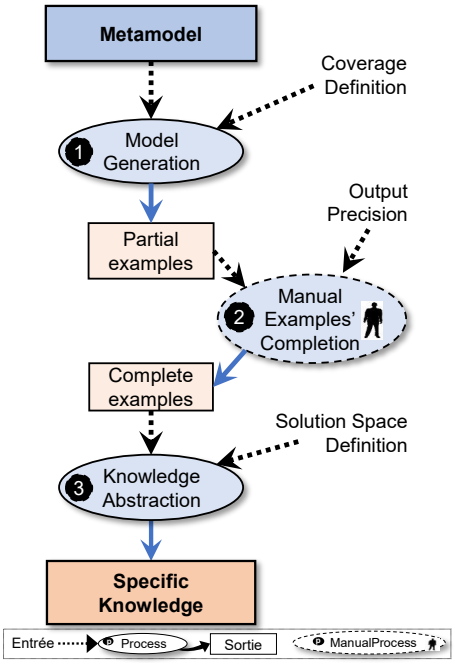


Figure 6.1. Thorough learning from example process in MDE

We consider the process of automatic learning from examples as a three steps process as illustrated in Fig. 6.1.

Schematically, the three steps are:

(1) **Example generation:** using metamodel definition and based on criteria such as coverage and size, a representative sample of input models must be provided to feed the learning algorithm (See Batot *et al.* (Batot et al, 2016) for more details).

(2) **Examples' completion:** input models must be completed with their respective output. This step must be executed manually. Forming complete examples is when the actual knowledge is injected. For example, model transformation input examples must be completed with their expected target model. Or, input model examples intended to induce wellformedness rules must be completed with their respective validity to the targeted domain of application.

(3) **Knowledge abstraction:** the problem is formulated as an optimization problem and genetic programming helps in finding near optimal solution which fall within a specified acceptable tolerance. The search for solutions is guided by a set of examples (*i.e.,* couple $(input, output)$) previously generated and completed. Using genetic programming to empirically search for solutions has been found to be (empirically) effective.

Definition 9. *An **MDE artifact** is an executable system that takes a model as input and produces a model as output (see Fig. 6.2).*

2.2. Concept of (Partial) Example

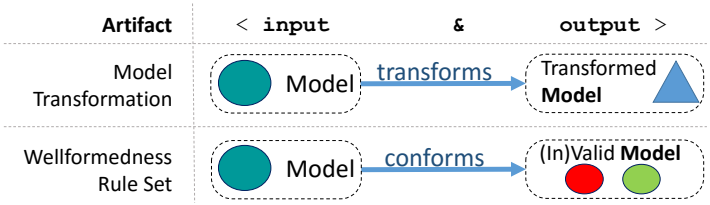


Figure 6.2. Structure of an example

Generally, MDE model management artifacts follow the same pattern: they take a model as input and output another model. For example, a model transformation defines the rules to be followed to produce, from a model conforming to the source metamodel, an "equivalent" model conforming to the target metamodel. More concretely, to learn a model transformation that transforms a UML-CD diagram into an RDBS diagram, an application example consists of a pair of models in the form: <UML-CD Diagram, RDBS Diagram>. If, on the other hand, the artifact to be learned is a set of wellformedness rules describing an intended application space, an example will be formed of 1) a model conforming to the metamodel defining the modeling space, and 2) an assessment validating or invalidating this model as part of the targeted sub-space (more on this example in next subsection).

Definition 10. *As illustrated in Fig. 6.2, **an example** of application of a given task consists of a pair of models: one is an input of the task; and the other corresponds to the output produced by the task when executed on that particular input.*

2.3. Consequences on automation

Definition 11. ***Quality of examples** is the representativeness of their input models over the input problem space.*

To properly capture the behavior of a task, we need to illustrate through a finite set of carefully selected models what result it is expected to produce. To supply a learning algorithm is then to find a set of examples that will be representative of the different possible cases that this task will potentially have to deal with.

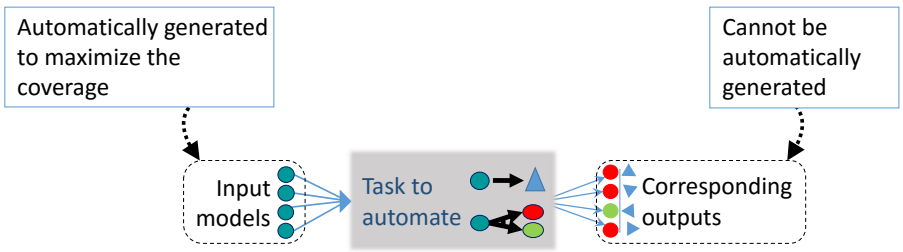


Figure 6.3. Disentangling problem space from solution space

Generating examples amounts to producing pairs of models. However, we have seen that in these couples, the output part is dependent on the input part and can only be completed manually. Fig. 6.3 illustrates these notions. However, the consequence of a notion relating to the examples as defined above is the decoupling of the problem space and the solution space. Thus, the metamodel defining the problem space can be used to generate the input parts of the examples independently of any considerations other than its size and representativeness.

If it is obvious that representativeness is essential, maintaining a reasonable size is crucial to allow a human actor to complete them. Thus, the input parts of the examples capturing the problem space can be generated automatically by using an approximation of their representativeness such as the coverage of the problem space. The coverage, as stated by Fleurey *et al.* (Fleurey et al, 2009) is the proportion of the metamodel that have been instantiated at least once to form the set of models.

The purpose of learning is to produce an artifact that reproduces this completion and generalizes it to a maximum of the possible inputs. We assume that qualification in terms of representativeness of the inputs influences the power of generalization of the tasks learned.

2.4. Search Space Reduction

The space of possible solutions of the learning problem is made of all possible instances the solutions' language enables to express. In other words, if a solution is written in a Language L, the solution space is composed of all possible grammatically correct formulations of that language. If the language L is Java, it will be all possible Java programs. If the language L defines model transformations (*e.g.,* ATL[2]), the solution space will be the union of all conforming instances, *i.e.,* all possible formulations of ATL that manipulate valid elements of the input and output metamodels. We call this space the space of Potential solutions —

[2] `http://www.eclipse.org/atl`

it contains every artifact well-formed and conform to the application domain. Among them, the valid solutions must be found.

This potential space remains nonetheless generally infinite and might be reduced in some cases. If the language L is ATL, a more Pragmatic solution space would consist of all solutions manipulating constructs of the metamodel instantiated in the examples (*e.g.,* the footprint of the transformation) rather than all of them. In the case of the Java language, reducing the scope of possible classes and methods to the ones of a certain API may help speed up the exploration of solutions. In the same fashion, using patterns, or templates, of higher order may as well augment the granularity of the search space and reduce the cost of its exploration. We depict an example of Pragmatic space reduction in Section 3.2 where OCL patterns are used to assist the learning of WFRs. The rationale behind is that OCL patterns instantiation, mapped on the constructs of the application domain, enables the expression of all *useful* formulations of the language.

Definition 12. ***Funneling toward a maneuverable solution space.***

- ***Conceptual Target:*** *here is the target we hope to find. It is an artefact doing the task we want.*
- ***Potential solution space:*** *all artifacts written in the chosen language could potentially solve our problem and automate the task. Unfortunately, we know that (most) artifacts written in the language will not fit our concern. An ATL transformation can find a solution to our problem (but most do not).*
- ***Pragmatic solution space:*** *we reduce the potential space by giving precision on the domain of application targeted. As an example, using transformation footprint or OCL patterns.*
- ***Valid solution space:*** *Finally, within the Pragmatic space, there are concrete solution artifacts. These artifacts solve all examples from the training set. Note that there might be more than one solution to our problem since high level languages allow syntactic and semantic synonyms.*

3. In Practice: Learning Wellformedness Rules from Examples

To illustrate our approach on learning complex artifacts from examples, let us consider the case of Wellformedness Rules (WFR) derivation from valid and invalid instance examples in an MDE context.

In this section, we start by proposing an improved methodology/algorithm to derive OCL constraints from examples and counter examples that addresses state-of-the-art limitations. Our methodology involves an enriched version of Faunes *et al.* (Faunes et al, 2013a) work using multi-objective fitness guidance coupled with recent advances in genetic programming (Batot and Sahraoui, 2018b). It is based on results on the use of WFR in MDE (Cadavid et al, 2015) for pragmatic space reduction.

3.1. Problem Statement

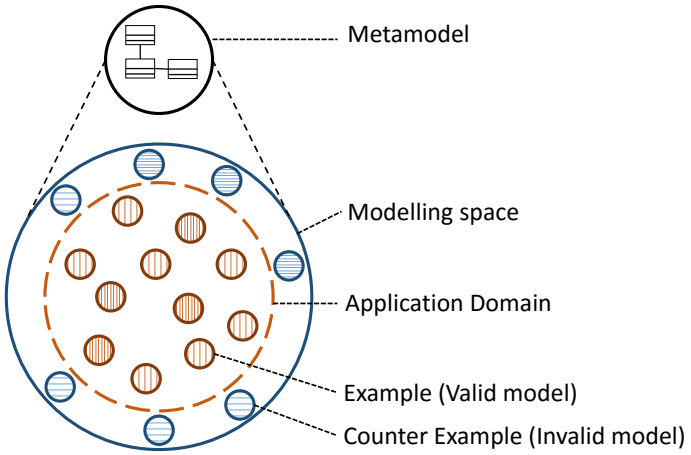


Figure 6.4. Modeling space, intended space, and (in)valid models

The definition of a modelling language begins with the edification of a metamodel representing the concepts or constructs to be instantiated, their attributes and the relationships they maintain with each other. The grammar of the language is thus defined roughly. Yet, all instances complying with these syntactic rules (metamodel compliant models) are not necessarily well formed w.r.t. the specific application domain targeted by the language (Cadavid et al, 2012).

Thus, as illustrated in Fig. 6.4, a metamodel defines a modeling space larger than the intended application domain. To better precise the modeling space, the addition of a WFRs set helps making the distinction between valid and invalid models.

For instance, consider a metamodel intended to capture the most basic elements to represent family trees. This metamodel, depicted in Fig. 6.5, has three concepts: `Person`, `Male` and `Female`. `Person` has a `mother` and a `father`, and possibly `children`. This basic structure will allow to instantiate any family tree. See Fig. 6.6 for examples of valid families. The first one exhibits a grandmother, her son and his two children; the second represents grandparents, their only son and his son; the bottom one shows a couple, their son and daughter and their respective children.

Nonetheless, if the metamodel captures properly all useful concepts, it does not avoid from building instances that break rules family trees are required to respect. As an example, consider a Person that references to itself through an instantiation of the feature `mother`. Here lies a hidden contract forbidding such construct – a Person cannot be its own mother (or father). To prevent these situations, it is possible to express in MDE WFRs with, for example, the Object Constraint Language (OCL[3]). Listing 6.1 shows a constraint written in OCL that forbids the instantiation of `own-mother` pattern. In such a constraint, the context defines on which class invariants apply. In this case, invariants involve mother and father references and refrain them from pointing to the context instance.

OCL allows to write complex rules by use of, among others, collection and set operators and functions.

However, the inherent complexity of OCL makes it a burden to write rules manually, especially for domain experts who are not familiar with formal languages. To circumvent this issue, Faunes *et al.* have shown feasible to learn some types of constraints from examples (valid models) and counter examples (invalid models) with respect to the targeted application domain (Faunes et al, 2013a).

[3]http://www.omg.org/spec/OCL/

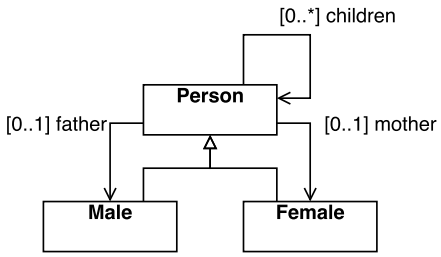


Figure 6.5. A type graph for Family DSML

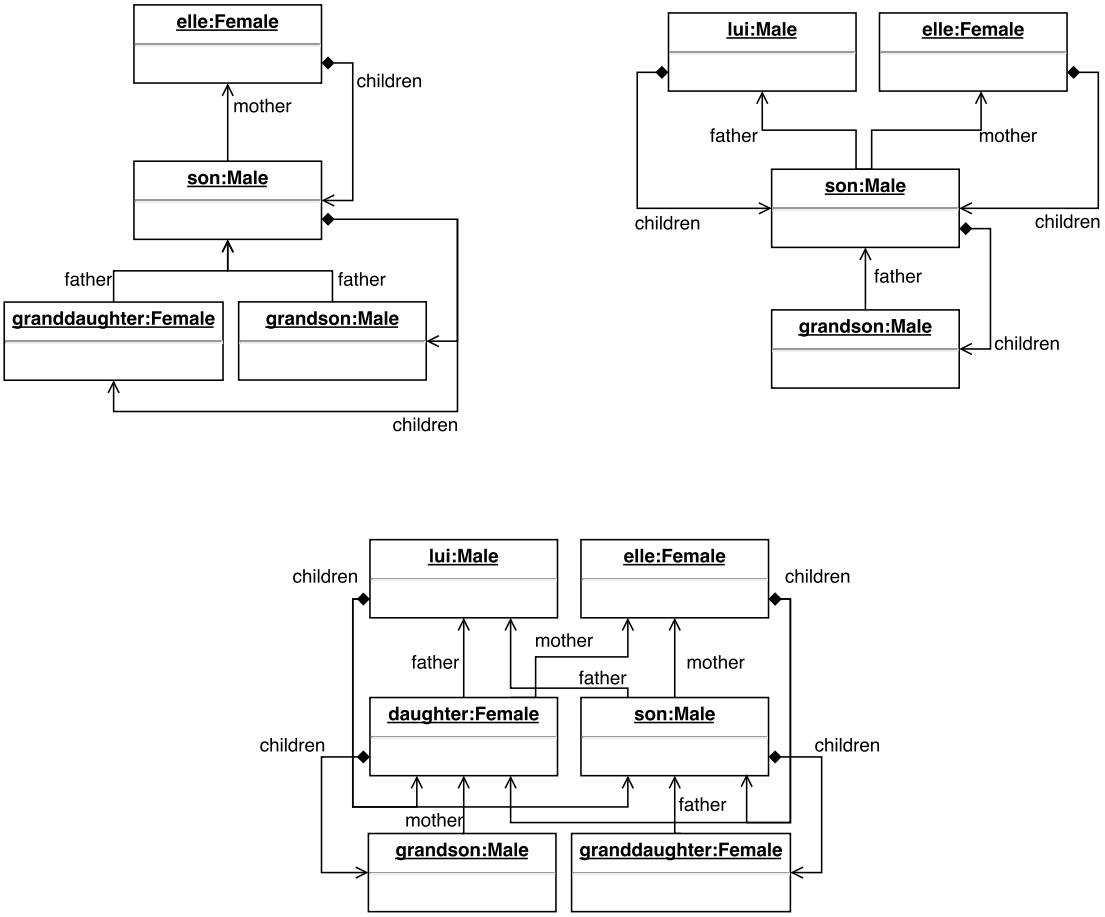


Figure 6.6. Examples of valid instances of the Family DSML

Listing 6.1. Wellformedness rule 'not-own-mother/father'

```
Context Person:
inv not-own-mother:
self.mother <> self
inv not-own-father:
self.father <> self
```

3.2. Space of Possible Solutions

The solution targeted by the learning process is a set of rules that differentiate automatically between valid and invalid models and draw the limit of the intent domain. But how to define the space within which to search for a good solution? Roughly speaking, and independently from its differentiation accuracy, any set of OCL constraints that can be written using the metamodel constructs is a potential solution. Such sets might contain simple constraints such as one testing that a person cannot be her own mother (see Listing 6.1).

Others can be more sophisticated like the constraint in Listing 6.2 which forbids a person to be her own ancestor, and which involves a graph closure operation. The set of all potential solutions is then very large, possibly infinite.

Although very large in theory, Cadavid shows in his PhD thesis (Cadavid Gómez, 2012) that the space of possible solutions can be reduced dramatically after studying dozens of metamodels at work in both academe and industry. He found that there exist 21 OCL patterns that could be combined to express all possible WFRs in the studied metamodels independently from the considered metamodel, i.e., patterns expressed at the meta-meta-level (MOF).

More precisely, 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. Fig. 6.7 gives the description of the *AcyclicReference* pattern.

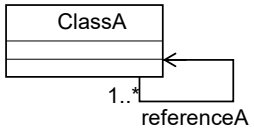
Coming back to our family-tree metamodel, the WFR of Listing 6.2 can be instantiated with two instances of the pattern A1 (see Fig. 6.7) as depicted in Fig. 6.8. This WFR is obtained by combining two instances of *AcyclicReference* pattern with different parameters. One involves `mother` reference, the other `father` reference.

The idea of exploiting some Cadavid's patterns to learn WFRs was first coined by Faunes *et al.* (Faunes et al, 2013a). Instead of searching for all possible OCL constraints, they focused on instances of some patterns. However, they did not exploit all the 21 patterns and considered only those dealing with reference cardinalities.

Listing 6.2. Wellformedness rule 'ancestry-bares-no-loop'

```
Context Person:
inv ancestry-bares-no-loop:
self.mother->closure()->excludes(self)
and
self.father->closure()->excludes(self)
```

- **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 reference the operation closure, which can only be invoked on collections.



- **OCL Expression Template:**

```
context ClassA inv AcyclicReference : referenceA
  ->closure(iterator: ClassA | iterator.referenceA)
  ->excludes( self )
```

- **Parameters:** ClassA, referenceA

Figure 6.7. MOF structure of pattern A1:*AcyclicReference*

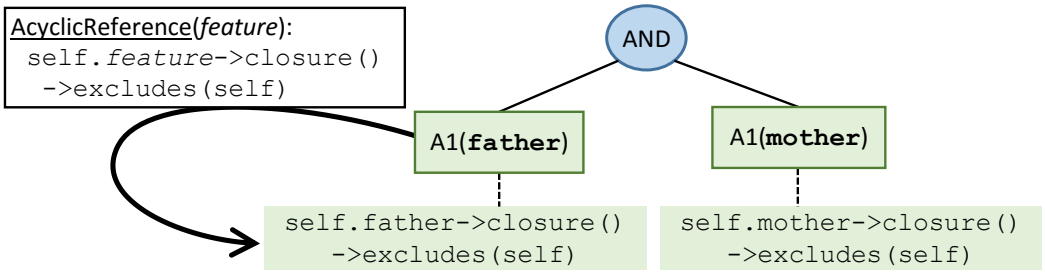


Figure 6.8. Abstract syntax tree of Wellformedness rule 'ancestry-bares-no-loop' with instantiations of pattern `AcyclicReference`

3.3. Need for Multi-Objective Search

Until now, we have explained how to make the learning search space tractable by considering only pattern instances. Let us now discuss how to guide the exploration of this space.

The presence of examples of valid and invalid models offers a convenient support for a semantic guidance of the search space. Thus, the first objective of the search is to maximize the number of examples a solution correctly classifies as valid/invalid.

Like programs, many OCL constraints sets may classify the model examples correctly. Some of these solutions can be verbose and include tautology constraints and constraint components. In GP, this phenomenon is known as *bloating* (or *code growth*) (Luke and Panait, 2006). To help producing constraints sets that are concise and humanly understandable, we consider the minimization of constraint-set size as another objective for bloat control.

When searching for the best constraint sets, another potential side effect of the correctness objective is the premature convergence towards local optima. Indeed, solutions that correctly classify a majority of cases have better chances to be considered during the evolution. However, those that classify a few but uncommon examples (not classified by the majority of solutions) tend to be ignored and be lost during the evolution due to their low accuracy score. To circumvent this phenomenon, maintaining diversity during the search is another concern to consider.

In summary, three different concerns will guide the exploration of the pragmatic space.

- **Classification accuracy**: the more examples a solution classifies accurately the better;
- **Size consideration**: the shorter a solution the more it is legible and maintainable by a human end user;
- **Diversity concern**: giving more chances to solutions that accurately classify uncommon examples improves our chances to learn the optimal WFR set (Batot and Sahraoui, 2018b).

These three concerns are competing by nature with one another. It is convenient to use a multi-objective algorithm. Thus, WFR learning can be considered as an optimization process with two main objectives: correctness (w.r.t. examples) and size of solutions. Additionally, diversity should be maintained during the search. We propose, then, to solve the WFR learning problem using multi-objective genetic programming (see Fig. 6.9) with NSGA-II (Deb et al, 2000), a multi-objective algorithm, illustrated in Fig. 6.10.

3.4. Adaptation/Implementation

In this section, we describe our implementation of multi-objective GP and its adaptation to WFR learning. We first introduce Multi Objective Optimization Problem (MOOP) and Multi Objective Genetic Programming vocabulary, their characteristics and main concepts before detailing how the learning of WFRs can be adapted to this methodology.

3.4.1. *Introduction to Multi Objective Genetic Programming*

The force of genetic programming is that the algorithm explores empirically a solution space (potentially infinite), guided by heuristic objectives based on potential solutions' outputs (*i.e.,* their semantics).

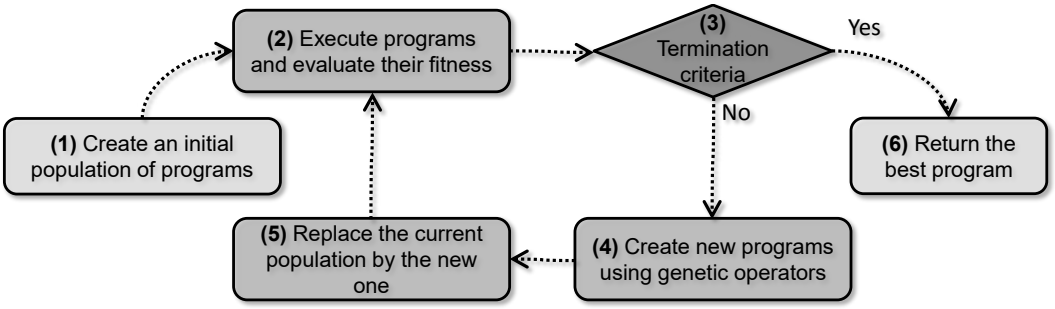


Figure 6.9. A typical genetic programming cycle

The most effective way to understand genetic programming is to look at the process sketched in Fig. 6.9. A first generation of solutions is created. Every entity in the ensuing population is executed on the input examples, and its output evaluated to measure its fitness. If a termination criterion is reached, the process ends and the solution is given to the user. Otherwise, the population is reproduced using genetic operators, and the new entities are executed and evaluated again. The process repeats until a termination criterion is reached. Adapting this process to a specific learning problem requires four steps: defining a solution representation for this problem; defining how new solutions are created from existing ones; defining how the fitness of a solution is measured; and defining a termination criterion. The adaptation should also take into account the multi-objective nature of our problem.

As we model the learning of WFRs as a multi-objective problem, let us give some basic definitions about MOOP.

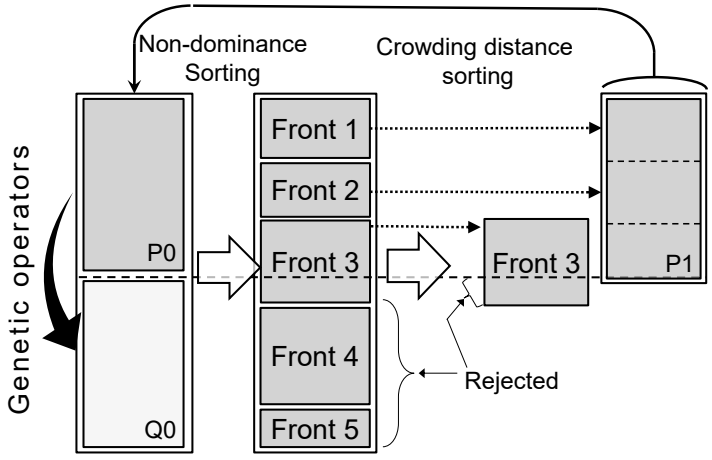


Figure 6.10. Non-Sorting Genetic Algorithm (NSGA-II) (Deb et al, 2000)

Definition 13. *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 14. ***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 $f_m(x) \geq f_m(x^*)$ and 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 15. ***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 16. ***Pareto optimal set:** For a MOOP f(x), the Pareto optimal set is $P^* = \{x \in \Omega | \neg \exists x' \in \Omega, f(x') \preceq f(x)\}$.*

We detail in the next subsections the four steps required to adapt WFR learning to MOOP. We detail how solutions are encoded and created. Then, we show how genetic operators stir the so provided genetic material. The third step consists in defining a manner to evaluate among a set of solutions which ones dominate, based on different criteria. Finally, since there is no way to ensure the optimal solution exists, an end criterion must be defined to put an end to the evolution process.

3.4.2. *Solution Representation and Initial Population Creation*

A solution to our problem is *a set of OCL constraints*. In our metaphor, a set of constraints is a genetic entity, containing K genes (individual constraints like one of Fig. 6.8).

More generally, a constraint, as pictured in Fig. 6.11, is represented as a tree whose nodes are operators and leaves are instances of WFR patterns. Operators are of two different kinds: logical operators and quantifiers.

Definition 17. *Considered logical operator are NOT, OR, AND, and $IMPLIES$.*

- ***NOT:** Negation of the subtree*
- ***AND:** Logical* `and` *between the two subtrees*

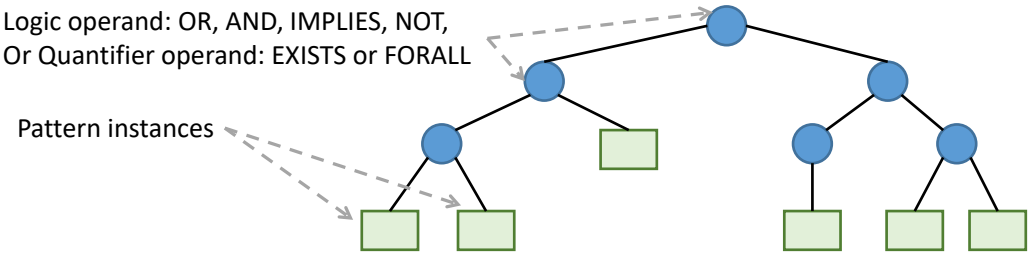


Figure 6.11. Abstract syntax tree of a wellformedness rule representation using patterns

- ***OR:*** *Logical* `or` *between the two sub trees*
- ***IMPLIES:*** *Logical* `implies` *between the two subtrees*

Considered quantifiers are forAll, and exists

- ***forAll:*** *Encapsulates a subtree in a* `forAll` *expression (See Listing 6.3)*
- ***exists:*** *Encapsulates a subtree in an* `exists` *expression (See Listing 6.3)*

Listing 6.3. Examples of using ForAll and Exists quantifiers

```
self.kids->forAll(Person k | k.mother = self)
self.kids->exists(Person k | k.mother = self)
```

The first step of the evolution consists in producing a random set of solutions (*i.e.,* constraint sets). To derive the initial generation G_0 of N solutions, we generate randomly N sets of k constraints, where k is randomly decided for each initial solution.

Each constraint set is built using randomly chosen operators and pattern instances. The average size of constraints (in terms of number of leaves) as well as the depth of their respective abstract syntax tree is parameterizable.

3.4.3. *Genetic Operators*

During the evolution, we derive new solutions from existing ones using two kinds of operators: **crossover** and **mutation**.

Crossover Operator

As illustrated in Fig. 6.12, the crossover operator uses a single cut-point. Each parent solution is divided into two constraint subsets using a randomly selected cut point. Then, the constraint subsets of the parent solutions are exchanged to form two new solutions. Note that the crossover does not alter the constraints as the cut point cannot be within a constraint tree.

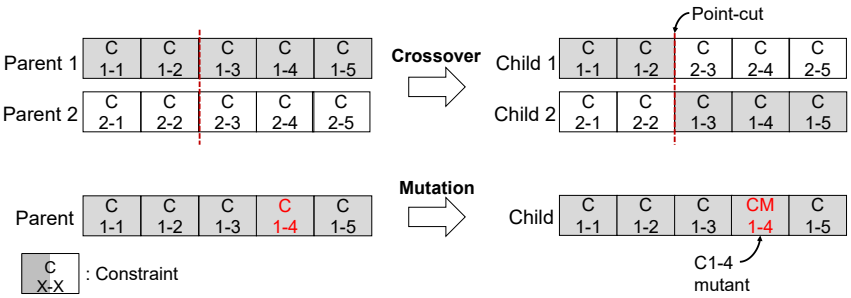


Figure 6.12. Crossover and mutation operators adapted to our framework

Mutation Operators We implemented different mutation operators, grouped in two categories.

- **Pattern mutation:** chooses randomly a leaf (*i.e.,* a pattern instance) in the constraint's tree and changes its parameters if there exist any valid alternative, or replace the pattern instance with a new one otherwise. See Fig. 6.13 for a sketch.
- **Logic mutations:** modifies the constraint's abstract tree by replacing an operator by another one, or by inserting a `NOT` operator or a quantifier. The logic mutations are applied as follows:

  (1) A node n_i is randomly selected from the constraint's tree ;

  (2) Then, either

  – n_i's operand is replaced by another compatible operand depending on the number of children. Binary operators AND, OR or $IMPLIES$ can replace each other. Similarly, unary operator NOT and quantifiers can be exchanged (Fig. 6.14); or

  – a quantifier or a NOT node is inserted between n_i and its parent (Fig. 6.15); or

  – n_i is replaced by a randomly created node (Fig. 6.16).

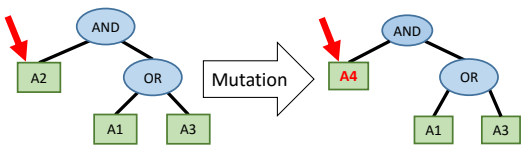


Figure 6.13. Pattern replacement

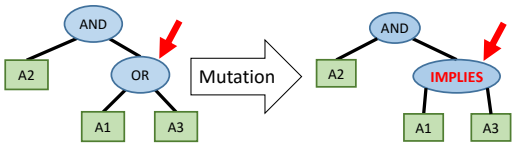


Figure 6.14. Node's operand replacement

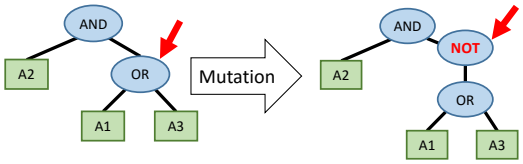


Figure 6.15. `NOT` node insertion

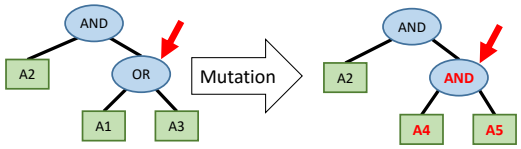


Figure 6.16. New node replacement

3.4.4. *Pruning*

Cabot *et al.* have shown how complex can become OCL and how the complexity can be reduced automatically (Cabot and Teniente, 2007). In our case, we used a pruning technique to limit the redundancy in abstract trees. When two sub trees are identical, we collapse them into one equivalent.

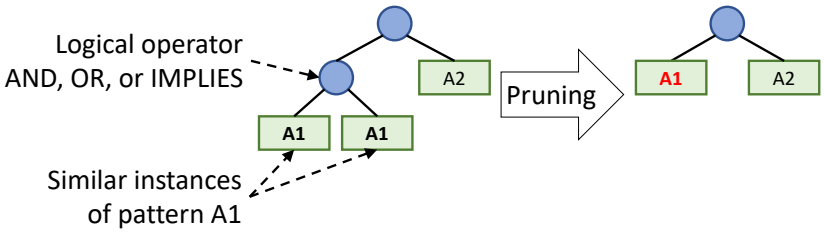


Figure 6.17. Pruning of similar subtrees

3.4.5. *Fitness*

As stated in Section 3.3, two objectives guide the search: the ratio of examples properly classified by a solution; and the constraint size of the solution. Additionally, a diversity measure is taken into consideration as well. To foster efficiency and generalizability of the

solutions, we embodied it in the *crowding distance measurement* of NSGA-II. See Batot *et al.* (Batot and Sahraoui, 2018b) for details.

3.4.6. *Termination Criterion*

We do not know, a priori, if a perfect solution exists, neither if it would be at reach. The termination criterion is thus set to a certain amount of iterations or a percentage of examples accurately treated (algorithm parameter).

3.5. Conclusion: On the Importance of the Quality of Partial Examples

The overall learning methodology discussed in Section 2 shows how subordinate the generalizability of solutions is to the representativeness of examples.

Yet, the only measure used to assess representativeness is the coverage of the examples with respect to the metamodel. We showed that this coverage considers the input parts of examples only. In the next section, we will see to which degree common coverage measurement impacts the quality of solutions and to which degree solutions comply with expectations.

4. Empirical Evaluation: Example Sets Characterization

In this section, we study the relationship between different characteristics of example sets and the inductive power they embody. We carry out such a goal by mean of a case study: the learning of WFR from examples and counter-examples in three distinct scenarios. We propose to investigate this relationship with three research questions targeting different levels of precision.

After sanity check, we question whether the **solutions are accurate** and test their generalizability on examples that were not used during learning. Then, we look more thoroughly at results to see if the **solutions are well-formed** and, later if they **convey the expected semantics**. Well-formedness evaluation implies a quantification of the distance between a solution and the oracle (*i.e.,* the expected solution) on a syntactic level. This will be our second research question. Finally, a semantic comparison between solutions and oracle is processed requiring a manual qualitative investigation.

Formally, our research questions are:

- **RQ0: Are the obtained results attributable to the approach itself or to the volume of explored solutions?** A SBSE sanity check consists in comparing results produced by our approach with the ones produced by a random search.
- **RQ1: Is the number of examples used to train the algorithm a factor influencing the power of differentiation of solutions?**
  We want to know if solutions give the expected behavior (*i.e.,* if they differentiate between models as expected) and if they are generalizable (*i.e.,* to an extended set of examples).
- **RQ2: Does the coverage of an example set influences the accuracy of learning solutions in terms of metamodel elements manipulated?**
  We then want to know if solutions are well-formed, if they manipulate the same metamodel constructs as the oracle.
- **RQ3: Considering the (20) better runs of all configurations, is our approach able to find the exact expected rules?**
  We finally manually inspect in depth the solutions: are they what we expect?

4.1. Experimental Setting

In this section, we introduce the material we used to process our empirical evaluation. We used different metamodels showing different levels of complexity. These metamodels are completed with the expected well-formedness rule sets. These expected set (oracle) give us a ground truth to compare with during the experiment. We give details about these elements and then depict the variables used: three independent variables to characterize the example sets; three dependent variables to grasp variations in solutions accuracy. Finally, we precise the parameters of the algorithm and how we tackle with its non-deterministic nature.

4.1.1. *Metamodels and Oracle*

In order to mitigate the influence of a metamodel specific structure on the learning, we selected three metamodels that demonstrate different levels of structure complexity and require diverse OCL constraints sets:

- `FamilyTree`: an excerpt has been introduced as our running example. This metamodel defines a schema for representing family structures. It has been used as an

illustrative example in various publications in the software modelling research literature, such as (Gogolla et al, 2015; Burgueno et al, 2015). It is composed of 4 classes, namely `Person`, `Male`, `Female`, and `License`. It has 6 features, of which 3 are references.

- `Statemachine`: a traditional example in software engineering. Composed of 11 classes and 10 features, of which 7 are references, it models vertice types and transitions of a state machine.
- `ProjectManager`: the most complex metamodel. It is composed of 5 classes and 23 features, of which 11 are references. It captures relations and features required in project management.

In a real-life scenario, an expert would differentiate between valid and invalid models. However for the need of our experiment, we use a semi-real scenario, in which we simulate the expert by the existing WFR rules. More concretely, we took the obvious well-formedneww rules in the case of the metamodel `FamilyTree`; from existing work for `Statemachine`(see (Cadavid Gómez, 2012)); and, from an online repository for the metamodel `ProjectManager` (described in (Hassam et al, 2010)). We used these WFR to also complete the examples for the learning algorithm.

For each metamodel, WFRs are:

- `FamilyTree`: three rules avoiding any loop in the ancestry of a person.

```
context Person
inv not-own-mother : this.mother->closure()->excludes(self)
inv not-own-father : this.father->closure()->excludes(self)
inv not-own-kid :    this.kids->closure()->excludes(self)
```

Listing 6.4. Oracle for `FamilyTree`

- `Statemachine`: eight rules constraining the number of incoming/outgoing transitions a vertex has depending on its type: `Initial`, `Final`, `Fork`, `Join`, or `Choice`.

```
context Initial
inv init_in : incoming->size() = 0

context Final
inv final_in : outgoing->size() = 0
```

```
context Choice
inv choice_in : incoming->size() > 1
inv choice_out : outgoing->size() > 1

context Fork
inv fork_in : incoming->size() = 1
inv fork_out : outgoing->size() > 1

context Join
inv join_out : outgoing->size() = 1
inv join_in : incoming->size() > 1
```

Listing 6.5. Oracle for `Statemachine`

- `ProjectManager`: seven rules of different kinds (*e.g.,* if an employee exists in a `Project`, there must be at least a project manager in that project).

```
context Employee
inv OfficeNumbDIFFoffAreaCode :
self.officeNumber <> self.officeAreaCode

context Project
inv AtLeastTwoProjectManagers:
self.department->forAll(d | d.employees->exists(e | e.
isProjectManager))

context Department
inv UniqueId:
self.projects->forAll(p | p.ident->size() <= 1)

context Employee
inv MaxTwoProject:
self.employer->size() <= 2

context BankAccount
```

```
inv BalancePositif:
self.balance > 0

context Department
inv DepartManager :
self.employees->size() = 0
or
self.employees->exists(e | e.oclIsKindOf(Manager))

context Employee
inv isManager:
self.oclIsKindOf(Manager) implies self.salary > 1500
```

Listing 6.6. Oracle for `ProjectManager`

4.1.2. *Examples*

Examples are made of a couple of models. Yet, as discussed in Section 2.2, we consider variations in the characteristics of input models only. We varied their size and coverage to test if these factors influence the learning process. To provide with diverse input models, we used a previous tool (Batot et al, 2016). More precisely, for each metamodel, sets of 5, 10, 15, 20, 25, 30, 35, and 40 examples were used (eight sets), with a balanced proportion of valid and invalid models. We created sets using two configurations:

- Randomly generating each set: eight sets of models are formed *independently*.
- *Incrementally* augmenting with (five) new examples an initial set of (five) examples until a maximum (40) is reached.

The rationale behind such a distinction is an attempt to account for the chance factor happening when using a new set of models. To diminished its effect on our results, we will compare both configurations.

In addition to these training sets, we provided another assortment of 100 models for each metamodel (with 50 valid and 50 invalid models) to compute the power to differentiate between models of solutions.

4.1.3. *Variables*

Fig. 6.18 sketches the general organization of the empirical study and highlight the elements that are involved in the variables. A training set of examples is required as input, together with the metamodel describing their input part. We are interested in measuring the size of the examples and their representativeness with respect to the metamodel (see Fig. 6.18, #1 and #2). From these, the learning process finds a solution, which precision is assessed by comparing its outputs with the provided oracle (see Fig. 6.18, #3). As mentioned earlier, the oracle represents the ground truth. Note that, examples forming the evaluation set were not used for training.

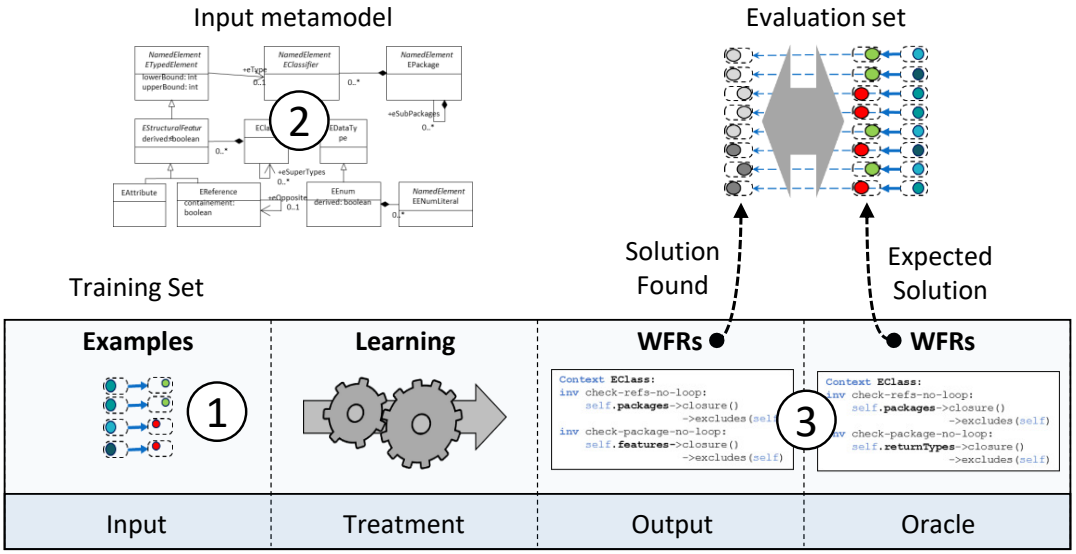


Figure 6.18. Empirical study general organization

We choose the following variables as a means to quantify our answers to the previous research questions.

Independent and Mitigating Variables. The main independent variable in our study is the coverage (COV) of the input part of the examples over the constructs of their metamodel. In addition to the coverage, we consider the size (SIZE), which is defined in terms of the number of examples in the training sets. Moreover, we assemble the sets in two different ways. A first alternative is to generate models sets of different sizes independently from one another. For instance, a set with 10 models has no common models with a set of 5 models. In the second alternative, models sets are augmented by adding new models to sets of smaller

sizes. Indeed, a set of 5 models is included in the set of 10 models, which in turn is included in the set of 15 models, and so on and so forth.

In a nutshell, we characterize our training example sets with three independent variables:

- **Size** (SIZE): number of models forming the training set;
- **Size increase kind** (KIND): a binary variable indicating if the models sets were generated independently ou incrementally;
- **Coverage** (COV): percentage of metamodel constructs found instantiated in at least one of the input models. For more details about the this metric, please refer to Fleurey et al (2009) for details.

Dependent variables. Dependent variables are factors that may be influenced by variations in the independent variables. We define three dependent variables corresponding to three levels of precision for the solutions.

Differentiating Rate (DIFF) First, we measure the *differentiating power* (DIFF) of a solution. To do so, we compare its output with the oracle for the validation set of 100 examples. DIFF is the percentage of examples of the *validation set* that the solution classifies accordingly to the oracle.

Solution Accuracy (ACCU) Then, in order to account for *solution accuracy* (ACCU), we extend our investigation to the syntax of solutions. We want to investigate if, with comparable DIFF, solutions may show differences in their structure. We address here the overwhelming power of expression of both OCL and models language. We ensure that a high DIFF does not hide a bias in the examples used. Indeed, back to our running example, it was a matter of finding a set of well-formedness rules able to distinguish between the well- and ill-formed examples of families. We know, for the sake of the experiment, that the targeted rules aim at avoiding loops in ancestry. However, considering the case where all ill-formed examples have persons under 6 years of `age` and well-formed examples only above that age. A rule that distinguishes models as invalid if they have an instance of `Person` whose `age` is less than 6 would have a perfect DIFF. Thus, in this configuration the feature `age` interferes in the differentiation. To measure the deviation from what was expected (a rule that deals with the feature `mother` and not `age`), we propose to compare the elements of the metamodel manipulated by the rules forming the solution with those manipulated by the oracle. Fig. 6.19 illustrates a situation where two elements coincide between the oracle

and the solution (in blue) and one element differs (in red). To compute this comparison, we extract all the elements manipulated by the oracle on one side, and those manipulated by the solution on the other side.

We then compute the following variables and metrics.

- SO = number of elements (of the metamodel) present in Solution and Oracle
- !SO = number of elements present in Oracle but not in Solution
- S!O = number of elements present in Solution but not in Oracle

$$Recall(S) = \frac{SO}{SO+!SO}$$
$$Precision(S) = \frac{SO}{SO+S!O}$$

We use the FMeasure as the ACCUracy of a solution: $ACCU = FMeasure(S)$

$$FMeasure(S) = 2 \times \frac{Precision(S) \times Recall(S)}{Precision(S) + Recall(S)}$$

Solution Relevance (RELE) Finally, we investigate the actual *relevance* (RELE) of solutions by performing a manual analysis. We classify a constraint c into one of the for following predicates:

(1) *Exact(c)* = true if the constraint is semantically equivalent to one of the oracle's constraints;

(2) *Relaxed(c)* = true if the constraint is more general or restrictive, but remains close to one of the oracle's constraint ;

(3) *Implicit(c)* = true if the constraint is implicitly included in the design of the metamodel (*i.e.,* redundant to some cardinalities); and

(4) *NoMatch(c)* = true if the constraint has nothing in common with any of the oracle's constraints

Finally, $RELE_E$ is the *exact* relevance, and $RELE_R$ the *relaxed* relevance, of a rule r. Formally:

$$RELE_E(r) = \sum_{true} Exact(r)$$
$$RELE_R(r) = \sum_{true} Exact(r) + \sum_{true} Implicit(r) + \sum_{true} Relaxed(r)$$

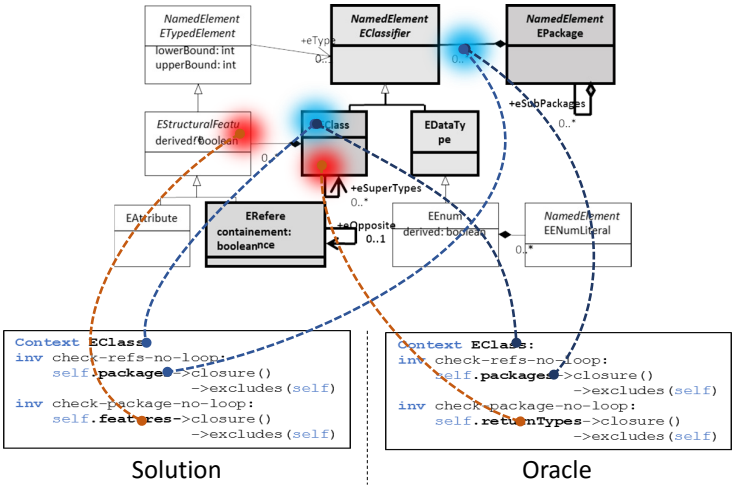


Figure 6.19. Solution accuracy measurement

Algorithm Parameters.

Genetic Parameters. Runs of the genetic program were all made of 3000 generations and 50 programs. Crossover probability was set to 0.9 and mutation probability to 0.35. The probability of both kinds of mutation was set to 0.5. The learning process was fully automated and did not involve any human intervention. It stopped either after achieving a 100% DIFF or completing 3000 iterations.

Tackling Indeterminism. Evolutionary algorithms are probabilistic by nature. Thence, identical settings may lead to different solutions. To ensure the stability of our algorithm, we ran the experiment 15 times for each configuration (except for answer RQ0 where we ran both configurations 30 times).

Multi-Objective Concern. An execution of a multi-objective algorithm yields a set of solutions called the Pareto-optimal front, in which solution cannot be ordered according to all the objectives. During the experiment, when the end criterion is reached, there was sometimes more than one solution in the Pareto. In order to evaluate our approach, we took the smallest among solutions showing the highest DIFF (*i.e.,* if there were more than one solution with 100% DIFF, we took the one with minimum SIZE).

4.1.4. *Validation Method*

To answer **RQ0**, we executed the learning algorithm on `ProjectManager` metamodel with the specified parameters. As we have a population of 50 solutions for each of the 3000 generations, we explore $50 * 3000 = 150000$ solutions. To have a fair comparison with random

search, we generated randomly the same amount of 150000 solutions. Then we compare the random vs learning solution groups using the Mann Witney statistical test in addition to measure the effect size with Cohen's d.

RQ1 looks at the relation between COV and DIFF and SIZE and DIFF. For the size, we study both configurations with independent and incremental model sets. For all the learning example configurations, we use 20 examples to learning and measure DIFF on the 100 validation examples. More precisely, this question comes to counting *how many test models were correctly differentiated*. To find a caesura, we compare results yielded by the different configurations using a t-test.

To answer **RQ2**, we look at the relation between independent variables (COV and SIZE) and the number of accurate metamodel elements manipulated by solutions (ACCU). This question comes to comparing *how many metamodel constructs are manipulated* by both the solution and oracle and if this number is influenced by COV and/or SIZE. We then focus on the relation between the 20 solutions showing best ACCUracy and COVerage of the examples used for their derivation.

Since **RQ3** requires an in-depth manual investigation of solutions, we studied 20 solutions that (i) show the best DIFFerentiating rate (RQ1), and (ii) use the maximum number of accurate metamodel elements (RQ2). From here, we investigate the relationship between DIFF and RELE. More precisely, this question comes to evaluating the *rate of exact/partial rules found* for solutions showing best DIFF and ACCU. In doing so, we ensure that a high fitness actually reveals a good solution.

4.2. Results

In this section, we present the results obtained and explain how they answer the research questions.

4.2.1. *RQ0: Are the obtained results attributable to the approach itself or to the volume of explored solutions?*

For both configurations, we used 20 examples to train the algorithm and measured ACC on the 100 validation examples. As shown in Table 6.1, with the simplest metamodel (`FamilyTree`), although our approach exhibits statistically-significant better results

(.98 compared .93 DIFF), the effect size is medium (0.74 Cohen's d). For the two other metamodels, the learning results are by far more higher than those of random search (0.5 DIFF for random search compared to 0.76 and 0.94 for automated learning). These results are statistically significant ($p-value < 0.001$) with a very strong effect size (5.38 and 7.35). We conclude then that ***the obtained results are attributable to the learning algorithm and not to the volume of explored solutions***.

4.2.2. *RQ1: Is the number of examples used to train the algorithm a factor influencing the power of differentiation of solutions?*

Fig. 6.20 and 6.21 show DIFF values (ord.) obtained when using different examples sizes (abs.) of the `ProjectManager` metamodel. They refer to sizes for respectively INDependent and INCremental model set generation.

We see that the greater is the number of examples, the higher is the DIFF value of solutions. Moreover, we observe a caesura between 10 and 15 models sets for both configurations. This difference between solutions learned with 10 examples and less, and solutions learned with 15 examples or more is statistically significant with $p-value < 0.01$. Comparing independent and incremental configurations does not bring much information. Finally, with more than 85% DIFF with sets of 40 examples, our approach is able to yield very interesting results.

We observed similar trends with the `Statemachine` metamodel, yet DIFF is slightly higher. Using the `FamilyTree` metamodel, 15 examples are enough to reach almost 100% DIFF. More examples do not add any substantive value.

In conclusion, ***the size of the learning example sets does influence the differentiation power of the learned solution, but this influence is stable after 15 examples.***

Table 6.1. Statistical comparison of DIFFerentiation power between random search and our approach on three WFR learning scenarios.

	Average DIFF Value		Mann Witney p-value	Effect Size Cohen's d
	Random	Our approach		
`ProjectManager`	0.5	0.76	<0.001	7.35
`Statemachine`	0.53	0.94	<0.001	5.38
`FamilyTree`	0.93	0.98	<0.001	0.74

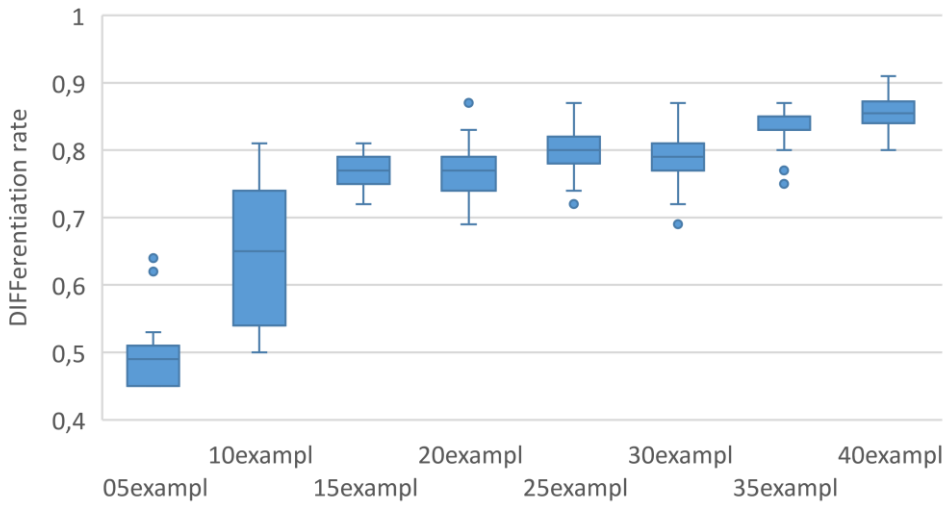


Figure 6.20. Box plot DIFFerentiation rate with metamodel `ProjectManager` with independent sets

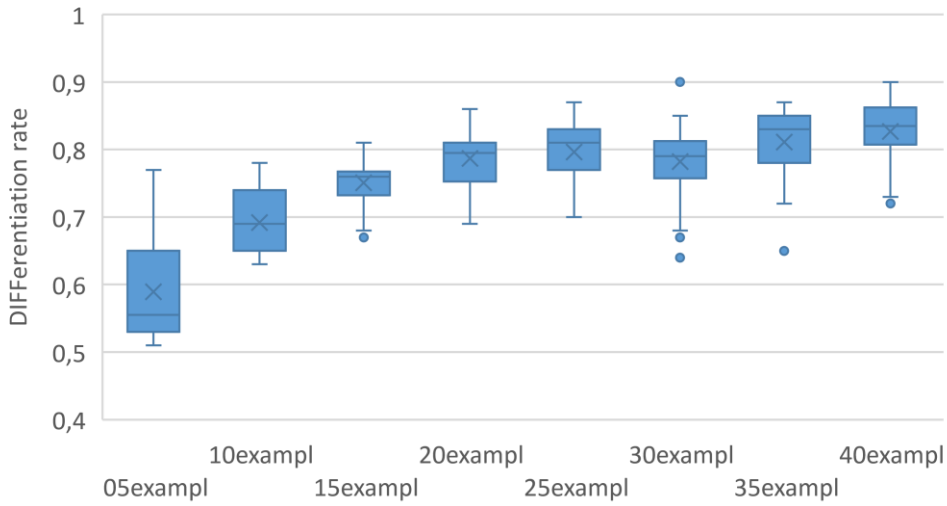


Figure 6.21. Box plot DIFFerentiation rate with metamodel `ProjectManager` with incremental set size augmentation

4.2.3. *RQ2: Does the coverage of an example set influence the accuracy of solution in terms of metamodel elements manipulated?*

Tables 6.2 shows the correlation between COV and ACCU, and SIZE and ACCU, for each of the three metamodels. We studied three configurations: considering all solutions

without consideration of the size variation (last column), or considering only independent (respectively incremental) size variation in column one (respectively column two).

At first glance, there is a striking difference between *independent* and *incremental* columns. The correlation between COV and ACCU, as well as the correlation between SIZE and ACCU, are stronger in the incremental configuration than in the independent configuration. When examples are provided incrementally, there is a common part of knowledge that transits from one configuration to the next. As expected, when the chance factor is diminished from one set size to the next, the correlation between COV and ACCU becomes stronger. Comparing between the second and third columns, line one, the incremental configuration yields an average but significant correlation (p-value < 0.004) whereas the independent configuration does not. Notwithstanding these results, there is a small yet significant correlation between COV and ACCU for the independant configiration for the two largest metamodels.

In addition, when looking at the relation between COV and ACCU of the best 20 solutions, strong and significant correlations arise for `ProjectManager`and `FamilyTree`metamodels (see Table 6.3). As a conclusion, ACCU seems to be a good heuristic for the accuracy of a set of constraints.

In conclusion, both size and coverage have generally an impact on the accuracy of the learned solutions. This impact is greater when the example sets are generated incrementally.

4.2.4. *RQ3: Considering the (20) better runs of all configurations, is our approach able to find the exact expected rules?*

Table 6.4 shows that, with averages of .505% for `ProjectManager`, .640% for `FamilyTree`, and .450% for `Statemachine`, our approach yields a significant amount of exact matches (column $RELE_E$). Moreover, if we look at the relaxed relevance ($RELE_R$), we see that with all metamodels, two thirds or more of the solutions are comparable to the oracle. This means that the algorithm is precise enough to find interesting rules. These results also indicate that when a solution is judged as accurate ($ACCU$), it is also relevant ($RELER_E$).

More precisely, here are some illustrations of the rules found, with their level of relevance.

(1) *Exact*: the constraint is semantically equivalent to one of the oracle's constraints. Example:

Table 6.2. Correlation between COV and ACCU, and SIZE and ACCU (p-value)

Metamodel		Independent	Incremental	All
`FamilyTree`	COV	.076 (.346)	.318 (.001)*	.076 (.346)
	SIZE	.012 (.880)	.292 (.004)*	.012 (.880)
`Statemachine`	COV	.195 (.003)*	.296 (.000)*	.209 (.346)
	SIZE	.238 (.000)*	.380 (.000)*	.292 (.000)*
`ProjectManager`	COV	.165 (.020)*	.334 (.000)*	.226 (.000)*
	SIZE	.248 (.000)*	.284 (.000)*	.260 (.000)*

Table 6.3. Correlation between COV and ACCU for the 20 best solutions

	Spearman	p-value
`ProjectManager`	.715	(.000)
`FamilyTree`	.669	(.000)
`Statemachine`	.106	(.047)

- Constraint from solution: "All BankAccount Departments have a Project Manager."

  ```
  Context BankAccount:
  self.department.employees
  ->exists(e:Employee | e.isProjectManager)
  ```

- Constraint from oracle: "All Departments have a Project Manager."

  ```
  Context Project:
  self.department->forAll(d:Department |
  d.employees->exists(e:Employee | e.isProjectManager))
  ```

Since all departments have, by definition, a bank account, the constraints are semantically equivalent.

Table 6.4. Average RELEvance of solutions

	$RELE_E$		$RELE_R$		$\sum_{true} NoMatch$		
	Average	Sigma	Average	Sigma	Average	Sigma	N
`ProjectManager`	.505	.25	.645	.20	.335	.20	20
`FamilyTree`	.640	.31	.650	.30	.350	.30	20
`Statemachine`	.450	.30	.815	.21	.160	.19	20
ALL	.532	.30	.703	.25	.282	.25	60

(2) *Relaxed*: the constraint is more general or restrictive, but remains close to one of the oracle's constraint.

Example:

- Constraint from solution: "A BankAccount owner's projects is from a department where there is at least one Employee that is a ProjectManager"

```
Context BankAccount:
self.person->forAll(e1:Employee |
e1.projects.department.employees->exists(e2:Employee |
e2.isProjectManager))
```

- Constraint from oracle: "All Departments have a Project Manager."

```
Context Project:
self.department->forAll(d:Department |
d.employees->exists(e:Employee |
e.isProjectManager))
```

The oracle rule checks that a project has a PorjectManager, the second ensures that a department (with projects) has at least a ProjectManager. This constraint is a *relaxed version* of the constraint expected.

(3) *Implicit* (*i.e.,* Found during evaluation): the constraint is implicit with regard to cardinalities of the typegraph.

Example: "A Department's bank account is related to its department."

```
Context Department:
self.bankAccounts->forAll(b:BankAccount |
b.department = self))
```

By definition, a Department has a reference bankAccounts (cardinality set to zero or many), which opposite is itself (cardinality set to one). This constraint is thus implicit (or redundant) to the design of the metamodel.

As a conclusion, we found a moderate but significant correlation between accuracy and size, and the quality of the solutions. We showed the soundness of our estimation with a manual investigation supported with statistical analysis. Solutions with high fitness are subject to contain one or more of the expected rules.

4.3. Threats to Validity

In our experiment, we considered only the problem of well-formedness rule learning. Yet, since the learning process remains much alike with other artifacts, we conjecture our methodology will apply for them as well and will ease its replication. Other experiments are, however, necessary to study how the results vary from one artifact to another. We are currently considering the learning of other artifacts such as model transformations and refactorings to prevent the mono-operation bias. Similarly, we are also considering the use of other size and coverage measures, as well as other ways of producing the oracle to handle the mono-method bias.

As we have seen, coverage is a key factor in the success of this investigation. Notwithstanding its predominance in the field, the coverage we use (Fleurey et al, 2009), has shown its limitation since it does not address in depth the multi-dimensionality of the instantiation of metamodels (Baudry et al, 2010). Recent studies showed a new interest and a potential in a new kind of coverage measurement (Semeráth et al, 2018b). It would be of great interest to replicate our experiment with such definitions. In this direction, the deepening of coverage includes a third important characteristic to example sets, diversity as described in (Ferdjoukh et al, 2017). Also, since the studies about coverage remain generic to the structure of the language and do not account for specific domain concerns, it might be valuable to further investigate the comparison of coverage evaluation between executions using examples built *ad hoc*, and executions whose examples are picked from real cases.

5. Related Work

Our contribution intersects two different research fields: Example-Based learning of MDE artifacts, and example and use case automatic generation. In the remainder of this section, we discuss some of the existing work in those fields.

5.1. Learning from examples

Learning from examples is a relatively new field (Bąk et al, 2013) which finds its root in the idea of a generative perspective on programming (Czarnecki et al, 1997) using SBSE (Harman and Jones, 2001).

Early work on model transformation learning was aimed at abstracting mappings between two metamodels starting from examples of source and target models (Balogh and Varró, 2009; Wimmer et al, 2007). These mappings were then transformed into transformation programs as done by Saada *et al.* (Saada et al, 2012). Kessentini *et al.* (Kessentini et al, 2012a) learn model transformations from examples 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.* (Faunes et al, 2013b) and Baki *et al.* (Baki et al, 2014; Baki and Sahraoui, 2016), learn directly the code of transformations from the application examples. Kessentini *et al.* also use examples to detect and fix defects (Kessentini et al, 2011b). Finally, examples are used to assist in modeling and metamodelling activities (Zayan et al, 2014; López-Fernández et al, 2013, 2015). In addition, works on WFR learning are mentioned in the next section.

Other teams have built up methodologies to assist the co-evolution of artifacts. Kessentini *et al.* (Kessentini et al, 2016) co-evolve models when the metamodel is modified. Batot *et al.* (Batot et al, 2017) automatically adapt OCL constraint sets after a modification to a metamodel. Lopez-Fernandez *et al.* use examples to assist metamodel conception (López-Fernández et al, 2013, 2015). Kessentini *et al.* use examples to detect and fix defects (Kessentini et al, 2011b).

Despite numerous work in this field, one striking commonality between these works (and the ones mentioned in next subsection) is the lack of quality assessment of the example sets used to train algorithms. As a matter of fact, it is obvious that a dearth of representativeness in the set of examples used to train the algorithm will result in biased solutions. Yet to date, examples are either i) made *ad hoc* to illustrate authors' words, or ii) are of great size provided by the industry. A systematic and automated evaluation is absent from literature.

Though, we found one early work worth mentioning on this topic. Fleurey *et al.* worked on qualifying input test data (Fleurey et al, 2009). Yet, they do not formalize a structured definition of what an example (or test data in their case) is expected to be. Later, Semeráth *et al.* (Semeráth and Varró, 2018) developped a deeper evaluation of diversity between models to enhance the generation process.

5.2. Learning WFR from examples

Examples were used to learn wellformedness rules in two main studies by Faunes *et al.* (Faunes et al, 2013a) and Dang *et al.* (Dang and Cabot, 2014). The latter infers OCL constraints in a semi automatic process where users' feedback is considered in an iterative fashion. Users can discard useless or misleading examples and help in choosing which solution seems more appropriate.

The present study is a follow up of the work by Faunes *et al.* (Faunes et al, 2013a) that features the learning of WFRs from examples and counter examples. The rules they propose to learn are simple and the scalability of the approach is at stake. Also, they used partial knowledge of the oracle to provide examples. More precisely, they used the oracle's rules, one by one, to generate training examples. For each one of them, they generated one valid model and one invalid using the *Alloy* solver (Jackson, 2006). A fact that hinders the generalization power of the study since the authors injected a knowledge that a real-life context does not offer. It also prevents from comparing its results with others. Moreover, Faunes *et al.* used a single objective GP algorithm. Since then, authors have been calling for multi-objective solutions to account for a distinction between syntactic and semantic concerns and to favor generalizability of solutions (Vanneschi et al, 2014; Wu, 2016).

In this study we propose a multi-objective algorithm capable of producing complex rules and a framework for the evaluation of example training sets that does not rely on extra knowledge of the problem.

5.3. Examples and use case generation

Hao discusses the state-of-the-art of metamodel instance generation approaches in a literature review that describes the different algorithms used as well as the various fields of application (Wu et al, 2012). In short, studies address specific problems and it is difficult to compare between them — less to generalize. Yet, the approach we choose proposes 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 (Batot et al, 2016).

Since then, the most striking advance we found in the literature is the work by Semerath and Varro (Semeráth et al, 2018b). The study is again intended to address the generation of test cases for model transformation, but their definition of coverage goes well

beyond previous work. The consideration of the richness of instances' features by means of neighborhood shapes representation seems to align better than Fleurey's coverage with the multi-dimensional aspect of meta-features instantiation. We envision to use their tool to test our methodology with this new coverage evaluation.

6. Conclusion and Discussion

In this paper, we study the importance of characteristics of model example sets during the learning of artifacts to automate MDE tasks. To this end, we start by analyzing the problem of learning MDE artefacts from examples and describe a holistic methodology to solve it. Then, we illustrate this approach on the specific problem of learning well-formedness rules from examples and counter examples. In this context, we propose a multi-objective genetic algorithm able to progressively extract WFR sets in the form of OCL constraints that differentiate adequately between valid and invalid model examples. The most important contribution of the paper is an empirical study of the influence of example sets characteristics on the learning quality, performed on the WFR learning problem. The major findings of this study are as follows. Firstly, the results show that having more examples helps learning more accurate artefacts, but this impact gradually decreases after a certain number of examples. Then, we found that there are moderate but statistically significant correlations between example sets' coverage and size, and the learned artifacts quality. Finally, after a manual inspection, it appears that solutions with high syntactic accuracy are generally semantically relevant as compared to the expected solutions. Although the results of our study are encouraging, we believe that experimenting with other MDE learning problems such as model transformations and code generations is necessary to draw a complete picture on example sampling w.r.t. the automation of knowledge derivation. Such studies should also consider other example characteristics in addition to the coverage and size, or at least other measures of coverage.

From the learning methodology stand point, our approach, described in Fig. 6.1, can be improved in different ways. For the model generation phase, the descriptions of coverage and size are given as inputs. The way we developed our algorithm makes it easy to integrate other coverage and size definitions/metrics. For the manual completion phase, we do not provide tools to support the experts in producing the outputs of examples. We are working

currently on a collaborative and interactive visualization setting with a multi-touch table to explore the collaboration between software engineers and domain experts for the example completions. The goal of this work is to mitigate the difficulty to deal with large size models and the inherent complexity of producing task outputs for them. For the learning phase, we have ongoing work to promote solution diversity during learning to improve the learning results and speed up the convergence. Our solution was successfully tested on WFR learning, but not on the learning of other artifacts. Finally, until now, we only focused on the accuracy and the relevance of the learned artifacts. In the future, it is important to consider non-functional aspects during the learning and evaluation of automated artifacts. For example, in addition to have correct model transformation programs, we want to have maintainable, evolvable, and non error-prone programs. This can be done by considering other objectives in the genetic programming algorithm.

Chapter 7

Conclusion

1. Contributions

The contributions addressed in this thesis are listed below.

A generic framework for model-set selection for learning or testing Model-Driven Engineering tasks. In this framework, the model-set selection is reformulated as a multi-objective optimization problem. Two objectives guide the search: maximizing the portion of the problem space covered by models whilst minimizing the size of the set. The framework can be tailored to the learning or testing of a specific task by firstly expressing a specific coverage criterion, which will confer precision to the first optimization objective. More precisely, the coverage definition is a way to express (or tag) 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 assess our method by comparison to the state-of-the-art in the case of metamodel edification. Results are encouraging and show that a deeper study of coverage has been long due.

A significant improvement of one of the most widely spread algorithm for multi-objective optimization, NSGA-II. We encoded a measurement for the social diversity of solutions during a genetic programming run. The Social Semantic Diversity (SSD) of a solution increases proportionally to the number of examples solved and is offset by the frequency with which an example is solved by the populations individuals. This helps to adjust for the fact that some examples are more frequently solved in general. Therefore, SSD favors solutions solving corner cases. We implemented this idea in two manners. One as an extra objective, and one as an alternative crowding distance. The results are breathtaking: solutions gain a significant increase in generalisation power and computation is up to thirty

times faster. We look forward to implementing it again in other future works and to sharing it widely with the larger community.

A pragmatic characterization of examples and a formalization of the thorough learning process. We formalized the process of learning MDE artifacts in three steps: partial examples generation, examples completion, and execution of metaheuristic algorithm using examples as a search guidance. This formalization permits to distinguish between problem and solution space and allows the consideration of specific coverage definition. It also specifies where human interaction is required and when automation can arise.

We used this formalization to set up an empirical investigation of the characteristics of examples that are susceptible to impact learning. We evaluated the impact of coverage on the relevance of the solutions through a three levels in-depth comparison between solutions and oracle. We found that a small but significant correlation exists between the coverage of the examples and the quality of the solution in terms of precision, power of generalization, and relevance. We foresee this characterization to be a milestone to improve awareness on the importance of reliability of EB-MDE technologies and thus to favor a greater adoption of MDE in general.

Other contributions. Finally, this thesis has also been an opportunity to put together other contributions. Not directly related to the present investigation these satellite works share the overall same direction. A first work that helped me understand the key concepts behind MDE was a systematic mapping study of concrete model transformations (Batot et al, 2016), published at MODELSWARD16 (Hammoudi et al, 2016). Then, a growing expertise in MOF-OCL put me to co-author a heuristic-based recommendation system to assist co-evolution of metamodel and wellfromedness rules (Batot et al, 2017). This work has been published at MoDELS17 (OConner, 2017). And finally, my expertise in genetic programming has been put to work in co-authoring an article for the detection of temporal API usage pattern (Saied et al, 2018) published this year at GECCO18 (Aguirre and Takadama, 2018).

2. Limits and future work

2.1. Short term

The selection of model sets is based on a set of random models. We used the AtlanMod Instantiator[1]. This tool has been of great help to provide with random examples, yet we intend to compare our results with a more up-to-date generator like Viatra [2] by Semeráth et al (2018b). Similarly, the definition of coverage we used has been dethroned (to our assumption) by this same study. They present a coverage that evaluate in-depth the way features are instantiated. An interesting future work would be to replicate the experiment with this coverage measurement – and review the overall definition of coverage accordingly. This change to our framework would require the implementation of this coverage measurement as a new module and would replace Fleureys coverage when evaluating model sets.

Finally, in the same vein, our experiment with SSDM could be replicated with partial examples coming from Viatra instead of AltanMod Instantiator.

2.2. Middle term

As Dang and Cabot (2014) show, it is difficult if not impossible to capture the knowledge of an expert in one go. The learning process must be iterative and take into account the feedback of the user. This feedback should focus on the quality of the examples, and we offer in our first contribution to redefine the definition of coverage for that purpose. It must also help to address the relevance of the solutions provided by the algorithm. To consider the iterative nature of the process, a valuable future work will consist in developing a tool to assist users to decide whether to include or exclude items manipulated by a solution in the assessment of coverage for an upcoming iteration. We envision a graphical tool enabling this selection. The next iteration will not start from scratch, but from the existing set of examples, adjusted to the new requirements.

Another limitation to the generalization of our results lies in the fact that we have applied our approach only to cases of learning WFRs. To alleviate this bias, we plan to replicate the experiments with model transformations. We would thus see if the rather positive trend

[1] `https://github.com/atlanmod/mondo-atlzoo-benchmark/tree/master/fr.inria.atlanmod.instantiator`
[2] `https://www.eclipse.org/viatra/`

of the barriers mentioned by Baudry et al (2010) for testing transformations is expansible and if the unification of model generation for learning and testing MDE tasks can be pushed ahead.

One of the recurring problems with the use of metaheuristics is the enormous cognitive effort necessary to apprehend the evolution of populations in time and space. Our WFR learning tool offers a visualization interface adapted to GP in the form of a matrix showing comparative results between oracle and solutions. This helped us understand GP runs and we plan to engage in more work on the visualization of evolution to increase the awareness of users as well as of designers.

2.3. And beyond

As we mentioned in our third contribution, size and coverage attributes of example sets show a moderate yet significant correlation with solution quality. With the learning methodology clearly defined, we envision the study of other attributes. As an example, diversity will figure as a third characteristic to investigate. There exists different way to evaluate diversity, different tools to *show* it graphically as well, but its impact on the process of learning has not been studied directly. A very interesting experiment would be the evaluation of how size, coverage and diversity correlate, how much the quality of examples depends on them, and how these results will benefits the general understanding of the learning process. With the everyday polemic on biases appearing in main medias and the newly mentioned role that GP can play in the deep-learning race to automation (Wilson et al, 2018), we foresee an increasing interest on this thesis approach and contributions.

Our research, and the SBSE in general, is much happening *in vitro*. The example Based MDE Learning community lacks empirical large-scale studies. A concrete experiment involving experts on a real case from industry, would be an important contribution. The problem though is serious. With the rising abstraction level of MDE, a so called real world study would involve experts of different fields where time is (very) costly – when available. Nonetheless, with the abovementioned growing interest in GP and the broadening adoption of MDE, there is hope to find investors ready to support such studies. On this matter, a robust framework would appeal on potential funders.

3. Coda

MDE shows an incredible power of abstraction. Following its key principles, everything seems ready to be represented through models and digested in automatic transformation chains. Even maintaining the coherence between different models during software evolution seems accessible. That idealisation of MDE seems within reach.

Yet, MDE is not mature enough to be universally accepted and the community would nurture its notoriety, if it could assess the potential of its automation. Its most recent advances, including the automatic learning of very complex artifacts, are impressive and show encouraging results. Nonetheless, EB-MDE broader adoption requires a serious effort to scope the assessment of such technologies out of the MDE field only.

To ensure its ability to produce robust and generalizable solutions, it is necessary to have a reliable and recognized standard to qualify its input data: example sets. This will allow the edification of a common framework and to clarify the relation between examples characteristics and solutions relevance. In that sense, MDE will gain directly from this contribution.

Bibliography

[Aguirre and Takadama 2018] Aguirre, Hernán E. (Editor) ; Takadama, Keiki (Editor): *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. ACM, 2018

[Amaral and Mernik 2016] Amaral, Vasco ; Mernik, Marjan: Special issue on quality in model-driven engineering. In: *Software Quality Journal* 24 (2016), Sep, Nr. 3, p. 597–599. – ISSN 1573-1367

[Anand et al 2013] Anand, Saswat ; Burke, Edmund K. ; Chen, Tsong Y. ; Clark, John ; Cohen, Myra B. ; Grieskamp, Wolfgang ; Harman, Mark ; Harrold, Mary J. ; Mcminn, Phil: An Orchestrated Survey of Methodologies for Automated Software Test Case Generation. In: *Internation Journal on Software and Systems Modeling* 86 (2013), Nr. 8, p. 1978–2001

[Anastasakis et al 2007] Anastasakis, Kyriakosand ; Bordbar, Behzadand ; Georg, Geriand ; Ray, Indrakshi: *UML2Alloy: A Challenging Model Transformation*. p. 436–450. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, Springer Berlin Heidelberg, 2007

[Aranega et al 2015] Aranega, Vincent ; Mottu, Jean-Marie ; Etien, Anne ; Degueule, Thomas ; Baudry, Benoit ; Dekeyser, Jean-Luc: Towards an automation of the mutation analysis dedicated to model transformation. In: *Software Testing, Verification and Reliability* 25 (2015), Nr. 5-7, p. 653–683. – ISSN 1099-1689

[Babur et al 2018] Babur, Önder ; Cleophas, Loek ; Brand, Mark van den ; Tekinerdogan, Bedir ; Aksit, Mehmet: Models, More Models, and Then a Lot More. In: Seidl, Martina (Editor) ; Zschaler, Steffen (Editor): *Software Technologies: Applications and Foundations*. Cham : Springer International Publishing, 2018, p. 129–135. – ISBN 978-3-319-74730-9

[Baki and Sahraoui 2016] Baki, Islem ; Sahraoui, Houari: Multi-Step Learning and Adaptive Search for Learning Complex Model Transformations from Examples. In: *ACM Transaction on Software Engineering and Methodology* 25 (2016), Nr. 3, p. 20:1–20:37. – ISSN 1049-331X

[Baki et al 2014] Baki, Islem ; Sahraoui, Houari ; Cobbaert, Quentin ; Masson, Philippe ; Faunes, Martin: Learning Implicit and Explicit Control in Model Transformations by Example. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)* Volume 8767. Springer International Publishing, 2014, p. 636–652

[Balogh and Varró 2009] Balogh, Zoltán ; Varró, Dániel: Model transformation by example using inductive logic programming. In: *Internation Journal on Software and Systems Modeling* 8 (2009), Nr. 3, p. 347–364

[Batot 2015] Batot, Edouard: Generating Examples for Knowledge Abstraction in MDE: a Multi-Objective Framework. In: *Proceedings of the ACM Student Research Competition at MODELS 2015 co-located with the ACM/IEEE 18th International ConferenceMODELS 2015.*, 2015, p. 1–6

[Batot et al 2017] Batot, Edouard ; Kessentini, Wael ; Sahraoui, Houari A. ; Famelis, Michalis: Heuristic-Based Recommendation for Metamodel - OCL Coevolution. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, IEEE Computer Society, 2017, p. 210–220

[Batot and Sahraoui 2016] Batot, Edouard ; Sahraoui, Houari: A Generic Framework for Model-Set Selection for the Unification of Testing and Learning MDE Tasks. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, 2016

[Batot et al 2016] Batot, Edouard ; Sahraoui, Houari ; Syriani, Eugene ; Molins, Paul ; Sboui, Wael: Systematic Mapping Study of Model Transformations for Concrete Problems. In: *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, 2016, p. 176–183

[Batot and Sahraoui 2018a] Batot, Edouard ; Sahraoui, Houari A.: Characterizing Example Sets for Knowledge Derivation in Model-Driven Engineering, ACM, 2018. – Submitted Fall 2018. Under revision.

[Batot and Sahraoui 2018b] Batot, Edouard ; Sahraoui, Houari A.: Injecting Social Diversity in Multi-objective Genetic Programming: The Case of Model Well-Formedness Rule Learning. In: *Search-Based Software Engineering - 10th International Symposium, SSBSE 2018, Montpellier, France, September 8-9, 2018, Proceedings* Volume 11036, Springer, 2018, p. 166–181

[Baudry et al 2006] Baudry, Benoit ; Dinh-Trong, Trung ; Mottu, JeanMarie ; Simmonds, D. ; France, Robert ; Ghosh, Sudipto ; Fleurey, F. ; Traon, Yves: Model transformation testing challenges. In: *Proceedings of the ECMDA workshop on Integration of Model Driven Development and Model Driven Testing* (2006)

[Baudry et al 2010] Baudry, Benoit ; Ghosh, Sudipto ; Fleurey, Franck ; France, Robert ; Le Traon, Yves ; Mottu, Jean-Marie: Barriers to Systematic Model Transformation Testing. In: *Commun. ACM* 53 (2010), Nr. 6, p. 139–143

[Baudry and Monperrus 2015] Baudry, Benoit ; Monperrus, Martin: The Multiple Facets of Software Diversity: Recent Developments in Year 2000 and Beyond. In: *ACM Comput. Surv.* 48 (2015), Nr. 1, p. 16:1–16:26

[Bersano-Begey 1997] Bersano-Begey, Tommaso F.: Controlling Exploration, Diversity and Escaping Local Optima in GP: Adapting Weights of Training Sets to Model Resource Consumption. In: Koza, John R. (Editor): *Late Breaking Papers at the 1997 Genetic Programming Conference*, 1997, p. 7–10

[Bézivin 2005] Bézivin, Jean: On the unification power of models. In: *Internation Journal on Software and Systems Modeling* 4 (2005), Nr. 2, p. 171–188

[Bąk et al 2013] Bąk, Kacper ; Zayan, Dina ; Czarnecki, Krzysztof ; Antkiewicz, Michał ; Diskin, Zinovy ; Wąsowski, Andrzej ; Rayside, Derek: Example-driven Modeling: Model = Abstractions + Examples. In: *Proceedings of the International Conference on Software Engineering (ICSE)*, IEEE Press, 2013 (ICSE '13), p. 1273–1276

[Boussaïd et al 2013] Boussaïd, Ilhem ; Lepagnot, Julien ; Siarry, Patrick: A Survey on Optimization Metaheuristics. In: *Information Science* 237 (2013), July, p. 82–117. – ISSN 0020-0255

[Boussaïd et al 2017] Boussaïd, Ilhem ; Siarry, Patrick ; Ahmed-Nacer, Mohamed: A survey on search-based model-driven engineering. In: *Automated Software Engineering* 24 (2017), Jun, Nr. 2, p. 233–294. – ISSN 1573-7535

[Brambilla et al 2012] Brambilla, Marco ; Cabot, Jordi ; Wimmer, Manuel: Model-driven software engineering in practice. In: *Synthesis Lectures on Software Engineering* 1 (2012), Nr. 1, p. 1–182

[Brottier et al 2006] Brottier, Erwan ; Fleurey, Franck ; Steel, Jim ; Baudry, Benoit ; Traon, Yves L.: Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. In: *17th International Symposium on Software Reliability Engineering (ISSRE 2006), 7-10 November 2006, Raleigh, North Carolina, USA*, IEEE Computer Society, 2006, p. 85–94

[Burgueno et al 2015] Burgueno, Loli ; Troya, Javier ; Wimmer, Manuel ; Vallecillo, Antonio: Static Fault Localization in Model Transformations. In: *IEEE Transactions on Software Engineering* 41 (2015), Nr. 5, p. 490–506

[Burke et al 2004] Burke, Edmund K. ; Gustafson, Steven ; Kendall, Graham: Diversity in genetic programming: an analysis of measures and correlation with fitness. In: *IEEE Transactions on Evolutionary Computation* 8 (2004), Feb, Nr. 1, p. 47–62. – ISSN 1089-778X

[Cabot and Teniente 2007] Cabot, Jordi ; Teniente, Ernest: Transformation Techniques for OCL Constraints. In: *Science of Computer Programming* 68 (2007), October, Nr. 3, p. 152–168. – ISSN 0167-6423

[Cadavid et al 2012] Cadavid, Juan J. ; Baudry, Benoit ; Sahraoui, Houari A.: Searching the Boundaries of a Modeling Space to Test Metamodels. In: *Proceedings of the International Conference on Software Testing Verification and Validation (ICST)*, 2012, p. 131–140

[Cadavid et al 2015] Cadavid, Juan J. ; Combemale, Benoit ; Baudry, Benoit: An Analysis of Metamodeling Practices for MOF and OCL. In: *Computer Language System and Structure* 41 (2015), p. 42–65

[Cadavid Gómez 2012] Cadavid Gómez, Juan J.: *Assisting Precise Metamodeling*, Université Rennes 1, France, Ph.D. thesis, 2012

[Clarisó and Cabot 2018] Clarisó, Robert ; Cabot, Jordi: Fixing defects in integrity constraints via constraint mutation. (2018)

[Clarke et al 2003] Clarke, John ; Dolado, Jose J. ; Harman, Mark ; Hierons, Robert ; Jones, Bryan ; Lumkin, Mary ; Mitchell, Brian ; Mancoridis, Spiros ;

Rees, Kearton ; Roper, Marc ; Shepperd, Martin: Reformulating software engineering as a search problem. In: *IEE Proceedings - Software* 150 (2003), June, Nr. 3, p. 161–175. – ISSN 1462-5970

[Cuadrado et al 2012] Cuadrado, Jesús S. ; Lara, Juan de ; Guerra, Esther: Bottom-Up Meta-Modelling: An Interactive Approach. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)* Volume 7590, Springer, 2012, p. 3–19

[Czarnecki and Helsen 2006] Czarnecki, Krzysztof ; Helsen, Simon: Feature-based survey of model transformation approaches. In: *IBM Systems Journal* 45 (2006), Nr. 3, p. 621–646

[Czarnecki et al 1997] Czarnecki, Krzysztof ; Ulrich, Eisenecker ; Steyaert, Patrick: Beyond Objects: Generative Programming. In: *ECOOP'97 Workshop on Aspect-Oriented Programming*. Jyväskylä, Finland : Springer, 1997

[Dabhi and Chaudhary 2012] Dabhi, Vipul K. ; Chaudhary, Sanjay: A Survey on Techniques of Improving Generalization Ability of Genetic Programming Solutions. In: *CoRR* abs/1211.1119 (2012)

[Dang and Cabot 2014] Dang, Duc-Hanh ; Cabot, Jordi: On Automating Inference of OCL Constraints from Counterexamples and Examples. In: *Knowledge and Systems Engineering - Proceedings of the Sixth International Conference KSE 2014, Hanoi, Vietnam, 9-11 October 2014*, Springer Berlin Heidelberg, 2014, p. 219–231

[Darwin 1859] Darwin, Charles: *On the Origin of Species by Means of Natural Selection*. London : Murray, 1859. – or the Preservation of Favored Races in the Struggle for Life

[Deb et al 2000] Deb, Kalyanmoy ; Agrawal, Samir ; Pratap, Amrit ; Meyarivan, T.: A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimisation: NSGA-II. In: *Parallel Problem Solving from Nature - PPSN VI, 6th International Conference, 2000, Proceedings*, 2000, p. 849–858

[Ehrig et al 2006] Ehrig, Karsten ; Koster, JochenM. ; Taentzer, Gabriele ; Winkelmann, Jessica: Generating Instance Models from Meta Models. In: *Formal Methods for Open Object-Based Distributed Systems* Volume 4037. Springer Berlin Heidelberg, 2006, p. 156–170

[Ekárt and Németh 2000] Ekárt, Anikó ; Németh, S. Z.: A Metric for Genetic Programs and Fitness Sharing. In: Poli, Riccardo (Editor) ; Banzhaf, Wolfgang (Editor) ; Langdon, William B. (Editor) ; Miller, Julian (Editor) ; Nordin, Peter (Editor) ; Fogarty, Terence C. (Editor): *Genetic Programming*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2000, p. 259–270. – ISBN 978-3-540-46239-2

[Faunes et al 2013a] Faunes, Martin ; Cadavid, Juan ; Baudry, Benoit ; Sahraoui, Houari ; Combemale, Benoit: Automatically Searching for Metamodel Well-Formedness Rules in Examples and Counter-Examples. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)* Volume 8107. Springer Berlin Heidelberg, 2013, p. 187–202

[Faunes et al 2011] Faunes, Martin ; Kessentini, Marouane ; Sahraoui, Houari A.: Deriving high-level abstractions from legacy software using example-driven clustering. In: *Center for Advanced Studies on Collaborative Research, CASCON '11, Toronto, ON, Canada, November 7-10, 2011*, IBM / ACM, 2011, p. 188–199

[Faunes et al 2013b] Faunes, Martin ; Sahraoui, Houari ; Boukadoum, Mounir: Genetic-Programming Approach to Learn Model Transformation Rules from Examples. In: *Proceedings of the International Conference on Theory and Practice of Model Transformation (IC-TPMT)* Volume 7909. Springer Berlin Heidelberg, 2013, p. 17–32

[Faunes Carvallo 2012] Faunes Carvallo, Martin: *Improving automation in model-driven engineering using examples*, Université de Montréal, Département d'informatique et recherche opérationnelle, Ph.D. thesis, 2012. – 1 vol. (104 p.) p. – Ph.D. thesis, supervised by Sahraoui, Houari

[Fenton and Pfleeger 1998] Fenton, Norman E. ; Pfleeger, Shari L.: *Software Metrics: A Rigorous and Practical Approach*. 2nd. Boston, MA, USA : PWS Publishing Co., 1998. – ISBN 0534954251

[Ferdjoukh et al 2013] Ferdjoukh, Adel ; Baert, Anne-Elisabeth ; Chateau, Annie ; Coletta, Remi ; Nebut, Clémentine: A CSP Approach for Metamodel Instantiation. In: *Proceedings of the International Conference on Tools with Artificial Intelligence*, Springer, 2013, p. 1044–1051

[Ferdjoukh et al 2017] Ferdjoukh, Adel ; Galinier, Florian ; Bourreau, Eric ; Chateau, Annie ; Nebut, Clémentine: Measuring Differences To Compare Sets Of

Models And Improve Diversity In MDE. In: *ICSEA: International Conference on Software Engineering Advances*. Athenes, Greece : Springer, October 2017

[Finot et al 2013] Finot, Olivier ; Mottu, Jean-Marie ; Sunyé, Gerson ; Attiogbé, Christian: Partial Test Oracle in Model Transformation Testing. In: *Proceedings of the International Conference on Theory and Practice of Model Transformation (IC-TPMT)*, 2013, p. 189–204

[Fleurey et al 2009] Fleurey, Franck ; Baudry, Benoit ; Muller, Pierre-Alain ; Le Traon, Yves: Towards Dependable Model Transformations: Qualifying Input Test Data. In: *Internation Journal on Software and Systems Modeling* 8 (2009), Nr. 2, p. 185–203

[Fogel 1999] Fogel, Lawrence J.: *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. New York, NY, USA : John Wiley & Sons, Inc., 1999. – ISBN 0-471-33250-X

[Fortin and Parizeau 2013] Fortin, Félix-Antoine ; Parizeau, Marc: Revisiting the NSGA-II Crowding-distance Computation. In: *Proceedings of International Conference on Genetic and Evolutionary Computation*, ACM, 2013 (GECCO)

[Galinier et al 2016] Galinier, Florian ; Bourreau, Eric ; Château, Annie ; Ferdjoukh, Adel ; Nebut, Clémentine: Genetic Algorithm to Improve Diversity in MDE. In: *META: Metaheuristics and Nature Inspired Computing*. Marrakech, Morocco : Archive ouverte HAL, October 2016

[García-Magariño et al 2009] García-Magariño, Iván ; Gómez-Sanz, Jorge J. ; Fuentes-Fernández, Rubén: Model Transformation By-Example: An Algorithm for Generating Many-to-Many Transformation Rules in Several Model Transformation Languages. In: Paige, Richard F. (Editor): *Theory and Practice of Model Transformations*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 52–66. – ISBN 978-3-642-02408-5

[Giraldo et al 2014] Giraldo, Faber D. ; España, Sergio ; Pastor, Oscar: Analysing the concept of quality in model-driven engineering literature: A systematic review. In: *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, May 2014, p. 1–12. – ISSN 2151-1349

[Gogolla et al 2005] Gogolla, Martin ; Bohling, Jørn ; Richters, Mark: Validating UML and OCL models in USE by automatic snapshot generation. In: *Software & Systems*

Modeling 4 (2005), Nov, Nr. 4, p. 386–398. – ISSN 1619-1374

[Gogolla et al 2007] Gogolla, Martin ; Büttner, Fabian ; Richters, Mark: USE: A UML-based specification environment for validating UML and OCL. In: *Science of Computer Programming* 69 (2007), Nr. 1, p. 27 – 34. – Special issue on Experimental Software and Toolkits. – ISSN 0167-6423

[Gogolla et al 2015] Gogolla, Martin ; Vallecillo, Antonio ; Burgueno, Loli ; Hilken, Frank: Employing Classifying Terms for Testing Model Transformations. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, 2015, p. 312–321

[Goldberg 1989] Goldberg, David E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1989. – ISBN 0201157675

[Gonzalez and Cabot 2014] Gonzalez, Carlos A. ; Cabot, Jordi: Test Data Generation for Model Transformations Combining Partition and Constraint Analysis. In: *Proceedings of the International Conference on Theory and Practice of Model Transformation (IC-TPMT)* Volume 8568, 2014, p. 25–41

[González Pérez et al 2012] González Pérez, Carlos A. ; Buettner, Fabian ; Clarisó, Robert ; Cabot, Jordi: EMFtoCSP: A Tool for the Lightweight Verification of EMF Models. In: *Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, 2012

[Guerra 2012] Guerra, Esther: Specification-Driven Test Generation for Model Transformations. In: *Proceedings of the International Conference on Theory and Practice of Model Transformation (IC-TPMT)*, 2012, p. 40–55

[Hammoudi et al 2016] Hammoudi, Slimane (Editor) ; Pires, Luís F. (Editor) ; Selic, Bran (Editor) ; Desfray, Philippe (Editor): *MODELSWARD 2016 - Proceedings of the 4rd International Conference on Model-Driven Engineering and Software Development, Rome, Italy, 19-21 February, 2016*. SciTePress, 2016. – ISBN 978-989-758-168-7

[Harman et al 2015] Harman, Mark ; Jia, Yu ; Zhang, Yang: Achievements, Open Problems and Challenges for Search Based Software Testing. In: *Proceedings of the International Conference on Software Testing Verification and Validation (ICST)*, 2015, p. 1–12

[Harman and Jones 2001] Harman, Mark ; Jones, Bryan F.: Search-based software engineering. In: *Information & Software Technology* 43 (2001), Nr. 14, p. 833–839

[Harman et al 2012] Harman, Mark ; Mansouri, S. A. ; Zhang, Yuanyuan: Search-based Software Engineering: Trends, Techniques and Applications. In: *ACM Computing Surveys (CSUR)* 45 (2012), December, Nr. 1, p. 11:1–11:61. – ISSN 0360-0300

[Hassam et al 2010] Hassam, Kahina ; Sadou, Salah ; Fleurquin, Régis: Adapting ocl constraints after a refactoring of their model using an mde process. In: *9th ed. of the BElgian-NEtherlands software eVOLution seminar*, 2010, p. 16–27

[Hoffmann and Minas 2010] Hoffmann, Berthold ; Minas, Mark: Defining Models - Meta Models versus Graph Grammars. In: *ECEASST* 29 (2010)

[Holland 1992] Holland, John H.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1992. – ISBN 0262082136

[Hutchinson et al 2011a] Hutchinson, John ; Rouncefield, Mark ; Whittle, Jon: Model-driven Engineering Practices in Industry. In: *Proceedings of the 33rd International Conference on Software Engineering*. New York, NY, USA : ACM, 2011 (ICSE '11), p. 633–642. – ISBN 978-1-4503-0445-0

[Hutchinson et al 2011b] Hutchinson, John ; Whittle, Jon ; Rouncefield, Mark ; Kristoffersen, Steinar: Empirical Assessment of MDE in Industry. In: *Proceedings of the International Conference on Software Engineering (ICSE)*. New York, NY, USA : ACM, 2011 (ICSE '11), p. 471–480. – ISBN 978-1-4503-0445-0

[Jackson 2006] Jackson, Daniel: *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006. – ISBN 0262101149

[Jeanneret et al 2011] Jeanneret, Cedric ; Glinz, Martin ; Baudry, Benoit: Estimating footprints of model operations. In: *Proceedings of the International Conference on Software Engineering (ICSE)*, May 2011, p. 601–610

[Jia and Harman 2011] Jia, Yue ; Harman, Mark: An Analysis and Survey of the Development of Mutation Testing. In: *IEEE Transaction on Software Engineering* 37 (2011), Nr. 5, p. 649–678

[de Jong et al 2001] Jong, Edwin D. de ; Watson, Richard A. ; Pollack, Jordan B.: Reducing Bloat and Promoting Diversity Using Multi-objective Methods. In: *Proceedings of*

the 3rd Annual Conference on Genetic and Evolutionary Computation, 2001 (GECCO'01), p. 11–18

[Kessentini et al 2011a] Kessentini, Marouane ; Sahraoui, Houari ; Boukadoum, Mounir: Example-based model-transformation testing. In: *Automated Software Engineering* 18 (2011), Nr. 2, p. 199–224

[Kessentini et al 2012a] Kessentini, Marouane ; Sahraoui, Houari ; Boukadoum, Mounir ; Omar, Omar B.: Search-based model transformation by example. In: *Internation Journal on Software and Systems Modeling* 11 (2012), Nr. 2, p. 209–226

[Kessentini et al 2008] Kessentini, Marouane ; Sahraoui, Houari A. ; Boukadoum, Mounir: Model Transformation as an Optimization Problem. In: *Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings*, Springer, 2008, p. 159–173

[Kessentini et al 2012b] Kessentini, Marouanne ; Sahraoui, Houari ; Boukadoum, Mounir ; Omar, Omar B.: Search-based model transformation by example. In: *Internation Journal on Software and Systems Modeling* 11 (2012), Nr. 2, p. 209–226

[Kessentini et al 2011b] Kessentini, Marouenne ; Kessentini, Wael ; Sahraoui, Houari ; Boukadoum, Mounir ; Ouni, Ali: Design Defects Detection and Correction by Example. In: *Proceedings of the International Conference on Program Comprehension (ICPC)*, 2011, p. 81–90

[Kessentini et al 2016] Kessentini, Wael ; Sahraoui, Houari A. ; Wimmer, Manuel: Automated Metamodel/Model Co-evolution Using a Multi-objective Optimization Approach. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, Springer, 2016, p. 138–155. – URL `https://doi.org/10.1007/978-3-319-42061-5_9`

[Kleppe et al 2003] Kleppe, Anneke G. ; Warmer, Jos ; Bast, Wim: *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003. – ISBN 032119442X

[Koza 1992] Koza, John R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA : MIT Press, 1992. – ISBN 0-262-11170-5

[Kuhlmann and Gogolla 2012] Kuhlmann, Mirco ; Gogolla, Martin: From UML and OCL to Relational Logic and Back. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*. Berlin, Heidelberg : Springer-Verlag, 2012 (MODELS'12), p. 415–431. – ISBN 978-3-642-33665-2

[López-Fernández et al 2013] López-Fernández, Jesús J. ; Cuadrado, Jesus S. ; Guerra, Esther ; Lara, Juan de: Example-driven meta-model development. In: *Internation Journal on Software and Systems Modeling* (2013), p. 1–25

[López-Fernández et al 2015] López-Fernández, Jesús J. ; Guerra, Esther ; Lara, Juan de: Example-based Validation of Domain-specific Visual Languages. In: *Proceedings of the International Conference on Software Language Engineering (SLE)*, 2015 (SLE 2015), p. 101–112

[Luke 2013] Luke, Sean: *Essentials of Metaheuristics*. second. Lulu, 2013. – Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/

[Luke and Panait 2006] Luke, Sean ; Panait, Liviu: A Comparison of Bloat Control Methods for Genetic Programming. In: *Evol. Comput.* 14 (2006), September, Nr. 3, p. 309–344. – ISSN 1063-6560

[McPhee and Hopper 1999] McPhee, Nicholas F. ; Hopper, Nicholas J.: Analysis of genetic diversity through population history. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2* Morgan Kaufmann Publishers Inc. (event), 1999, p. 1112–1120

[McPhee et al 2008] McPhee, Nicholas F. ; Ohs, Brian ; Hutchison, Tyler: Semantic Building Blocks in Genetic Programming. In: *Genetic Programming*, Springer Berlin Heidelberg, 2008, p. 134–145

[Mens and Van Gorp 2006] Mens, Tom ; Van Gorp, Pieter: A Taxonomy of Model Transformation. In: *Electron. Notes Theor. Comput. Sci.* 152 (2006), March, p. 125–142. – ISSN 1571-0661

[Mohagheghi and Aagedal 2007] Mohagheghi, Parastoo ; Aagedal, Jan: Evaluating Quality in Model-Driven Engineering. In: *International Workshop on Modeling in Software Engineering (MISE'07: ICSE Workshop 2007)*, May 2007, p. 6–6. – ISSN 2156-7883

[Mohagheghi et al 2013] Mohagheghi, Parastoo ; Gilani, Wasif ; Stefanescu, Alin ; Fernandez, Miguel A. ; Nordmoen, Bjørn ; Fritzsche, Mathias: Where does model-driven engineering help? Experiences from three industrial cases. In: *Internation Journal on Software and Systems Modeling* 12 (2013), Nr. 3, p. 619–639

[Mokaddem et al 2018] Mokaddem, Chihab E. ; Sahraoui, Houari ; Syriani, Eugene: Recommending Model Refactoring Rules from Refactoring Examples. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, 2018

[Mottu et al 2012] Mottu, Jean-Marie ; Sen, Sagar ; Tisi, Massimo ; Cabot, Jordi: Static Analysis of Model Transformations for Effective Test Generation. In: *International Symp. on Software Reliability Engineering*, 2012, p. 291–300

[Murashkin et al 2013] Murashkin, Alexandr ; Antkiewicz, Michał ; Rayside, Derek ; Czarnecki, Krzysztof: Visualization and exploration of optimal variants in product line engineering. In: *International Software Product Line Conference* ACM (event), 2013, p. 111–115

[Mussbacher et al 2014] Mussbacher, Gunter ; Amyot, Daniel ; Breu, Ruth ; Bruel, Jean-Michel ; Cheng, Betty H. C. ; Collet, Philippe ; Combemale, Benoit ; France, Robert B. ; Heldal, Rogardt ; Hill, James ; Kienzle, Jörg ; Schöttle, Matthias ; Steimann, Friedrich ; Stikkolorum, Dave ; Whittle, Jon: The Relevance of Model-Driven Engineering Thirty Years from Now. In: Dingel, Juergen (Editor) ; Schulte, Wolfram (Editor) ; Ramos, Isidro (Editor) ; Abrahão, Silvia (Editor) ; Insfran, Emilio (Editor): *Model-Driven Engineering Languages and Systems*. Cham : Springer International Publishing, 2014, p. 183–200. – ISBN 978-3-319-11653-2

[OMG 2012] OMG: *OMG Object Constraint Language (OCL), Version 2.3.1*. 2012

[OMG 2013] OMG: *OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1*. June 2013

[Ostrand and Balcer 1988] Ostrand, Thomas J. ; Balcer, Marc J.: The Category-partition Method for Specifying and Generating Fuctional Tests. In: *Commun. ACM* 31 (1988), Nr. 6, p. 676–686

[OConner 2017] OConner, Lisa (Editor): *20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, Austin, TX, USA,*

September 17-22, 2017. IEEE Computer Society, 2017. – ISBN 978-1-5386-3492-9

[Paige et al 2016] Paige, Richard F. ; Matragkas, Nicholas ; Rose, Louis M.: Evolving models in Model-Driven Engineering: State-of-the-art and future challenges. In: *Internation Journal on Software and Systems Modeling* 111 (2016), p. 272 – 280. – ISSN 0164-1212

[Paige et al 2017] Paige, Richard F. ; Zolotas, Athanasios ; Kolovos, Dimitris: *The Changing Face of Model-Driven Engineering*. p. 103–118. In: Mazzara, Manuel (Editor) ; Meyer, Bertrand (Editor): *Present and Ulterior Software Engineering*. Cham : Springer International Publishing, 2017

[Popoola et al 2016] Popoola, Saheed ; Kolovos, Dimitrios S. ; Rodriguez, Horacio H.: EMG: A Domain-Specific Transformation Language for Synthetic Model Generation. In: Van Gorp, Pieter (Editor) ; Engels, Gregor (Editor): *Theory and Practice of Model Transformations*. Cham : Springer International Publishing, 2016, p. 36–51. – ISBN 978-3-319-42064-6

[Ryan 1994] Ryan, Conor: (1994)

[Saada et al 2012] Saada, Hajer ; Dolques, Xavier ; Huchard, Marianne ; Nebut, Clémentine ; Sahraoui, Houari A.: Generation of Operational Transformation Rules from Examples of Model Transformations. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, 2012, p. 546–561

[Sadilek and Weißleder 2008] Sadilek, Daniel A. ; Weissleder, Stephan: Testing Metamodels. In: *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*, Springer, 2008, p. 294–309

[Saied et al 2018] Saied, Mohamed A. ; Sahraoui, Houari A. ; Batot, Edouard ; Famelis, Michalis ; Talbot, Pierre-Olivier: Towards the automated recovery of complex temporal API-usage patterns. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, Springer, 2018, p. 1435–1442

[Schaffer 1985] Schaffer, J. D.: Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA : L. Erlbaum Associates Inc., 1985, p. 93–100. – ISBN

0-8058-0426-9

[Schmidt 2006] Schmidt, Douglas C.: Model-driven engineering. In: *IEEE Computer Society* 39 (2006), Nr. 2

[Selic 2003] Selic, Bran: The Pragmatics of Model-Driven Development. In: *IEEE Softw.* 20 (2003), September, Nr. 5, p. 19–25. – ISSN 0740-7459

[Selic 2012] Selic, Bran: What will it take? A view on adoption of model-based methods in practice. In: *Internation Journal on Software and Systems Modeling* 11 (2012), Nr. 4, p. 513–526

[Selim et al 2012] Selim, Gehan M. K. ; Cordy, James R. ; Dingel, Juergen: Model Transformation Testing: The State of the Art. In: *Workshop on the Analysis of Model Transformations*, 2012, p. 21–26

[Semeráth et al 2018a] Semeráth, Oszkár ; Nagy, András S. ; Varró, Dániel: A graph solver for the automated generation of consistent domain-specific models. In: *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Springer, 2018, p. 969–980

[Semeráth et al 2018b] Semeráth, Oszkár ; Nagy, András S. ; Varró, Dániel: A graph solver for the automated generation of consistent domain-specific models. In: *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Springer, 2018, p. 969–980

[Semeráth and Varró 2018] Semeráth, Oszkár ; Varró, Dániel: Iterative Generation of Diverse Models for Testing Specifications of DSL Tools. In: Russo, Alessandra (Editor) ; Schürr, Andy (Editor): *Fundamental Approaches to Software Engineering*, Springer International Publishing, 2018, p. 227–245

[Sen et al 2009] Sen, Sagar ; Baudry, Benoit ; Mottu, Jean-Marie: Automatic Model Generation Strategies for Model Transformation Testing. In: *Proceedings of the International Conference on Theory and Practice of Model Transformation (IC-TPMT)* Volume 5563. Springer Berlin Heidelberg, 2009, p. 148–164

[Sendall and Kozaczynski 2003] Sendall, Shane ; Kozaczynski, Wojtek: Model transformation: the heart and soul of model-driven software development. In: *IEEE Software* 20 (2003), Sept, Nr. 5, p. 42–45. – ISSN 0740-7459

[Soule and Foster 1998] Soule, Terence ; Foster, James A.: Effects of Code Growth and Parsimony Pressure on Populations in Genetic Programming. In: *Evolutionary Computation* 6 (1998), Nr. 4, p. 293–309

[Sparck Jones 1988] Sparck Jones, Karen: Document Retrieval Systems. In: Willett, Peter (Editor): *Document Retrieval Systems*. London, UK, UK : Taylor Graham Publishing, 1988, Chap. A Statistical Interpretation of Term Specificity and Its Application in Retrieval, p. 132–142. – ISBN 0-947568-21-2

[Steinberg et al 2009] Steinberg, David ; Budinsky, Frank ; Paternostro, Marcelo ; Merks, Ed: *EMF: Eclipse Modeling Framework 2.0*. 2nd. Addison-Wesley Professional, 2009. – ISBN 0321331885

[Strommer et al 2007] Strommer, Michael ; Murzek, Marion ; Wimmer, Manuel: Applying Model Transformation By-Example on Business Process Modeling Languages. In: Hainaut, Jean-Luc (Editor) ; Rundensteiner, Elke A. (Editor) ; Kirchberg, Markus (Editor) ; Bertolotto, Michela (Editor) ; Brochhausen, Mathias (Editor) ; Chen, Yi-Ping P. (Editor) ; Cherfi, Samira Si-Saïd (Editor) ; Doerr, Martin (Editor) ; Han, Hyoil (Editor) ; Hartmann, Sven (Editor) ; Parsons, Jeffrey (Editor) ; Poels, Geert (Editor) ; Rolland, Colette (Editor) ; Trujillo, Juan (Editor) ; Yu, Eric (Editor) ; Zimányie, Esteban (Editor): *Advances in Conceptual Modeling – Foundations and Applications*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 116–125. – ISBN 978-3-540-76292-8

[Strommer and Wimmer 2008] Strommer, Michael ; Wimmer, Manuel: A Framework for Model Transformation By-Example: Concepts and Tool Support. In: Paige, Richard F. (Editor) ; Meyer, Bertrand (Editor): *Objects, Components, Models and Patterns*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, p. 372–391. – ISBN 978-3-540-69824-1

[Sun and Gray 2009] Sun, Jules ; Gray, Jeff: Model Transformation by Demonstration. In: Schürr, Andy (Editor) ; Selic, Bran (Editor): *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, 2009, p. 712–726

[Umuhoza et al 2015] Umuhoza, Eric ; Ed-douibi, Hamza ; Brambilla, Marco ; Cabot, Jordi ; Bongio, Aldo: Automatic Code Generation for Cross-platform, Multi-device Mobile Apps: Some Reflections from an Industrial Experience. In: *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle*. New York, NY, USA : ACM,

2015 (MobileDeLi 2015), p. 37–44. – ISBN 978-1-4503-3906-3

[Utting et al 2012] Utting, Mark ; Pretschner, Alexander ; Legeard, Bruno: A Taxonomy of Model-based Testing Approaches. In: *Proceedings of the International Conference on System Testing Verification and Reliability (STVR)* 22 (2012), August, Nr. 5, p. 297–312

[Vanneschi et al 2014] Vanneschi, Leonardo ; Castelli, Mauro ; Silva, Sara: A Survey of Semantic Methods in Genetic Programming. In: *Genetic Programming and Evolvable Machines* 15 (2014), Nr. 2, p. 195–214. – ISSN 1389-2576

[Varró 2006] Varró, Dániel: Model Transformation by Example. In: *Proceedings of the International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, 2006, p. 410–424

[Whittle et al 2014] Whittle, Jon ; Hutchinson, John ; Rouncefield, Mark: The State of Practice in Model-Driven Engineering. In: *IEEE Software* 31 (2014), Nr. 3, p. 79–85

[Williams 2013] Williams, James R.: *A Novel Representation for Search-Based Model-Driven Engineering*, University of York, Department of Computer Science, Ph.D. thesis, 2013. – 1 vol. (263 p.) p. – PhD Thesis

[Wilson et al 2018] Wilson, Dennis G. ; Cussat-Blanc, Sylvain ; Luga, Hervé ; Miller, Julian F.: Evolving Simple Programs for Playing Atari Games. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA : ACM, 2018 (GECCO '18), p. 229–236. – ISBN 978-1-4503-5618-3

[Wimmer et al 2007] Wimmer, Manuel ; Strommer, Michael ; Kargl, Horst ; Kramler, Gerhard: Towards Model Transformation Generation By-Example. In: *40th Hawaii International Conference on Systems Science*, 2007, p. 285

[Wu 2016] Wu, Hao: Generating Metamodel Instances Satisfying Coverage Criteria via SMT Solving. In: *Proceedings of the International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2016, p. 40–51

[Wu et al 2012] Wu, Hao ; Monahan, Rosemary ; Power, James F.: Metamodel Instance Generation: A systematic literature review. In: *CoRR* abs/1211.6322 (2012)

[Wyns et al 2006] Wyns, Bart ; De Bruyne, Peter ; Boullart, Luc: Characterizing Diversity in Genetic Programming. In: *Proceedings of the 9th European Conference on*

Genetic Programming. Berlin, Heidelberg : Springer-Verlag, 2006 (EuroGP'06), p. 250–259. – ISBN 3-540-33143-3, 978-3-540-33143-8

[Zayan et al 2014] Zayan, Dina ; Antkiewicz, Michał ; Czarnecki, Krzysztof: Effects of Using Examples on Structural Model Comprehension: A Controlled Experiment. In: *Proceedings of the International Conference on Software Engineering (ICSE)*, ACM, 2014 (ICSE 2014), p. 955–966

[Zhang et al 2014] Zhang, Yuanyuan ; Harman, Mark ; Mansouri, Afshin: *The SBSE Repository: A repository and analysis of authors and research articles on Search Based Software Engineering*. 2014