A DSL for Model Mutation and its Applications to Different Domains

Pablo Gómez-Abajo

Universidad Autónoma de Madrid, Spain,
Pablo.GomezA@uam.es
http://www.miso.es

Abstract. Model-Driven Engineering (MDE) is a Software Engineering paradigm that focuses all phases of the software development process in models. Therefore, the automated manipulation of models is essential in MDE. While many Domain-Specific Languages (DSLs) exist to specify model transformation, model simulation, or code generation, there is a lack of DSLs to specify and apply model mutations. A model mutation is a kind of model manipulation that creates a set of variants (or mutants) of a seed model by the application of one or more mutation operators. Model mutation has many applications (e.g., model transformation testing, automated generation of exercises, verification of software testing, etc.). The objective of this thesis is the creation of a DSL for model mutation, and its application to different domains like education, model-based testing and search-based software engineering.

Keywords: Domain-Specific Languages, Model-Driven Engineering, Model Mutation, Education, Automated Generation of Exercises, Software Testing Verification, Evolutionary Algorithms

1 Problem

Model-Driven Engineering (MDE) [4,19] uses models in all phases of the software development process. Hence, model manipulation is a key activity in MDE, for which domain-specific languages (DSLs) particularly tailored to the transformation task are heavily used [14].

A *model mutation* is a kind of model manipulation that creates a set of variants (or *mutants*) of a seed model by the application of one or more *mutation operators*. Model mutation has many applications. For example, in model transformation testing [1], a transformation is represented as a model that is mutated to evaluate the efficacy of a test set. Such a test set may have been created by mutation of a set of input seed models. In education, a model representing a correct solution in a domain (like a class diagram, an automaton or an electronic circuit) is mutated to produce exercises (e.g., consisting in the identification of the errors injected by the mutations) that can automate the suggested answer [8,18].

There are some frameworks for model mutation, but they are specific for a language (e.g., logic formulae [11]) or domain (e.g., testing [1,2]); moreover, mutation operators are normally created using general-purpose programming languages that are not tailored to the definition and production of mutants. Hence, there is a lack of proposals facilitating the definition of mutation operators, applicable to arbitrary languages and applications. These would facilitate the creation of domain-specific mutation frameworks like the abovementioned ones by providing: high-level mutation primitives (e.g., for object creation or reference redirection) together with strategies for their customization; support for composition of mutation operators; handy integration with external applications (e.g., in education, search-based software engineering, model-based testing) through dedicated post-processors or compilation into a general-purpose language; and traceability of the applied mutations.

2 Related work

In this section, we review related works on the main line of related research: the application of mutation techniques to different contexts and domains.

Mutation is used in areas like model-based testing [13], model transformation testing [1], program testing [6,22], adaptive systems [2], embedded systems [3], evaluation of clone-detection algorithms [21], generation of large model sets [17], education [18], or evolutionary algorithms [12,15]. Most of these systems are built ad-hoc for a specific domain; therefore, a DSL for model mutation that is domain-independent would be helpful in automating their construction. Next, we review approaches related to model mutation in different domains.

The mutation framework in [1] is specific to model transformation testing. It provides a set of predefined mutation operators which are transformation-specific and are defined with the Kermeta general-purpose model management language[1]. As an alternative, a DSL that has mutation-specific primitives, and is not restricted to the mutation testing domain would be useful. Also for mutation testing, the language MuDeL [20] provides a description of mutation operators for grammar-based artefacts, typically programs. The mutation operators in [6] are specific to testing Android apps. We see again that a DSL for model mutation with further facilities to combine mutation operators, apply them several times, and discard mutants that are non-conforming to the meta-model, or duplicated ones, would facilitate these works. Mutation is also central in some search-based engineering approaches [10], especially in evolutionary algorithms, where problems are solved by generating a set of candidate solutions, which is iteratively improved by applying crossover and mutation operators. Candidate solutions are usually encoded as bit arrays, though some recent works [15] propose model-based approaches where the domain is expressed as a meta-model, the candidate solutions as models, and the mutation operators as model mutators.

3 Proposed solution

To facilitate the specification and creation of model mutations in a meta-model independent way, we propose a DSL called WODEL. Fig. 1 shows the architecture of our approach. First, the user provides a set of seed models conformant to a meta-model. Then, they use WODEL to define the desired mutation operators and their execution details, like how many mutations of each

[1] `http://www.kermeta.org/documents/`

type should be applied in each mutant, or their execution order. In addition, each WODEL program needs to declare the meta-model of the models to mutate, which can be any meta-model because WODEL is meta-model independent. This allows type-checking the program to ensure it only refers to valid meta-model types and properties, and allows checking that the result of the mutation is valid.

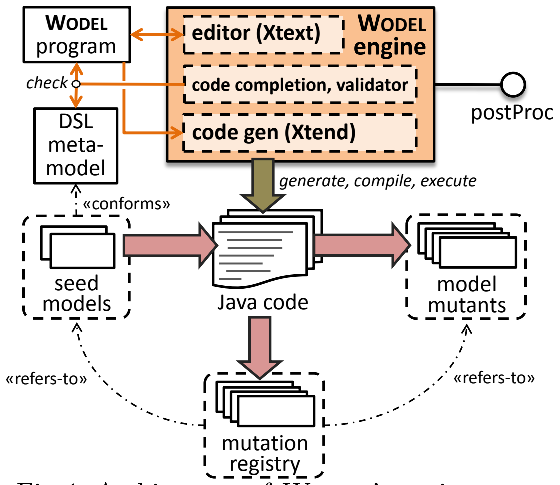

Fig. 1: Architecture of WODEL's environment

Executing a WODEL program produces mutants of the seed models. These are still valid models (i.e., they conform to the seed models' meta-model) as this is checked upon generating each mutant. If the mutant is not conforming to the meta-model, WODEL iterates the mutant generation until a maximum n number of times (that can be configured through the preferences page). The produced Java code, which is in charge of creating the mutants from the seed models as well as the registry of applied mutations, can be transparently executed from the WODEL IDE. The registry stores the applied mutations, along with references to the elements affected by the mutations in the seed model or in the mutants. This registry generation of the applied mutations is useful when we need to repeat the mutation process, or we have to generate natural language that describes the applied mutations. Finally, an optional post-processing step can be used to generate domain-specific artefacts for particular applications of the mutants. This will be provided by the WODEL environment through an extension point. This way, WODEL can also generate mutants in other formats different to EMF XMI (e.g., json, et.). We plan to extend WODEL to three different applications: automated generation of exercises; software testing verification; and search-based software engineering.

We plan to provide WODEL with correct and incorrect mutant generation: correct mutants are conforming to the same meta-model of the seed model, and incorrect mutants may violate some of its constraints (e.g., do not satisfy some association cardinality constraint). This set of correct and incorrect mutants will be useful in evolutionary computation (e.g., an incorrect mutant may have better fitness according to a given criterion than a correct one). WODEL will be able to distinguish the conforming mutants and the ones that are not conforming to the meta-model. Also, we will include mechanisms to identify duplicate mutants not only syntactically, but also semantically. These mechanisms ensure that the generated mutants will be unique and they are useful in the automated generation of exercises (e.g., we ensure that the selectable text options, or the set of diagrams, presented in an exercise correspond to different models), and also in software testing verification (e.g., we ensure that the mutants of a program to be verified are behaviourally different).

Wodel will provide mutation primitives: creation, deletion, edge redirection, cloning, etc.; and also model element selection strategies: random selection, and property-based selection (e.g., a selection with a fitness function as a property will be useful in evolutionary algorithms). We also plan to broaden Wodel with execution policies: parallel, sequential, distributed (e.g., two Wodel programs can be executed in parallel in different threads at the same time, so they generate different results which can be merged afterwards). Also, it will be useful to provide Wodel with libraries of reusable mutations for particular domains (e.g., a library of interesting mutations for the automated generation of exercises of a particular domain). We find it useful to provide Wodel with a registry generation of the applied mutations (e.g., to reverse a mutation, or for traceability reasons). It may also be beneficial to extend the mutations registry with additional behaviour: compacting the registry (e.g., it is irrelevant to store in the registry two mutations that cancel each other: one mutation creates an object and another one deletes it). The registry will also be useful for repeatability (e.g., it will be necessary in evolutionary computation, to be able to repeat the mutation process). It will be also interesting to extend this registry with some mechanisms to verbalize the applied mutations (e.g., verbalizing the applied mutations forward or backward will be useful to explain students how to do or undo the mutations in the automated generation of exercises, in order to create the correct answers that fix the mutant; or the wrong ones, that correspond to alternative mutants).

4 Preliminary work

We have developed an environment which allows the creation of Wodel programs and their compilation into Java, and can be extended with post-processor steps for particular applications. Wodel can also generate a registry of the applied mutations. We also have created the Wodel-Edu extension to Wodel, an application on the automated generation of test exercises.

Wodel programs have two parts. The first one declares the number of mutants to generate, the output folder, the seed models and their meta-model. The second part defines mutation operators and how many times they should be applied. Programs can also include a list of OCL constraints that all generated mutants should fulfil. Details can be checked in [8]. Listing 1 shows a simple Wodel program. Line 1 states that we want to generate 3 mutants in folder out, from the seed model evenBinary.fa. Line 2 indicates the meta-model of the seed model. These configuration parameters can be omitted, and Wodel will take default predefined values. These default predefined values can be configured in the editor's preference page. Lines 4–9 define three mutation operators: the first one (lines 5–6) selects randomly a final state, and makes it non-final; the second one (line 7) creates a new final state; and the last one (line 8) creates a new transition from the state selected in line 5 to the one created in line 7.

```
1 generate 3 mutants in "out/" from "evenBinary.fa"
2 metamodel "http://fa.com"
3
4 with commands {
5   s0 = modify one State where {isFinal = true}
6          with {reverse(isFinal)}
7   s1 = create State with {isFinal = true}
8   t0 = create Transition with {src = s0, tar = s1, symbol = one Symbol}
9 }
```

Listing 1: A simple WODEL program

WODEL-EDU is an extension to WODEL for the automated generation of test exercises. It provides four DSLs to configure the text and style of exercises (EDUTEST), how model elements should be graphically rendered (MODELDRAW), how to represent a model element textually (MODELTEXT), and how to represent an applied mutation textually (MUTATEXT). WODEL-EDU is also domain-independent and generates a web application with exercises for different domains (e.g., automata, class diagrams, electronic circuits, etc.). Currently, the generated exercises are of multiple response, but we plan to extend the framework to support more dynamic exercises, with a better interaction with the student. A preliminary evaluation of WODEL-EDU showed good results (the generated application can be accessed on-line at `http://www.wodel.eu`).

5 Expected contributions

WODEL will ease the creation of applications based on mutations by providing support for their definition, execution and traceability. In addition, we will develop the following three post-processing extensions:

1. Automated generation of exercises that can automate the suggested answer. This is our framework WODEL-EDU (see Section 4), which will be extended with gamification features, and interactive exercises.
2. Software testing verification. The source code of a program (represented as a model) will be mutated via a set of mutation operators. The quality of the test cases will be evaluated by checking if they detect the generated mutants. In addition, we will generate a library of WODEL encoded mutations. This library will apply to any general-purpose programming language.
3. Search-and model-based software engineering. In this approach, population-based search is used to optimize a problem. Problems are represented with models, and search is performed by mutating the models in the population. We can use a similar approach as MOMoT [7] to select the best mutants, but with the advantage that our language is more flexible to express mutations.

6 Plan for evaluation and validation

We will evaluate the expressivity of WODEL using in this DSL the interesting mutations both devised by us and found in the literature [9,16,18]. We will use the test exercises generated with Wodel-Edu in university courses about automata theory, electronic circuits, and others. We will use the software testing verification framework with real software projects, with the collaboration of the industry. We will also use the approach to test ATL model transformations, complementing the previous work of our group [5]. We will use the search-based engineering environment with the help of our colleagues of the AIDA[2] research group within our department, who are experts in this area.

7 Current status

We have developed a preliminary version of Wodel which supports 7 types of mutation primitives and 4 selection strategies, and composite mutations. We also have included the registry extension to store which mutations have been applied to each seed model at the mutants' generation. We have improved the Wodel DSL with new mutation primitives; conditional expressions for the specific selection of elements; and also the declaration of blocks inside Wodel programs, in order to generate mutants at different stages. These improvements were necessary for applying Wodel to the automated generation of exercises of the Wodel-Edu framework. Currently, Wodel-Edu supports three kinds of test exercises: alternative response, multiple diagram choice, and multiple emendation choice. The design and development of Wodel includes the evaluation of the expressivity of the DSL. The work should be finished by the middle of 2019, as it is shown in Fig. 2.

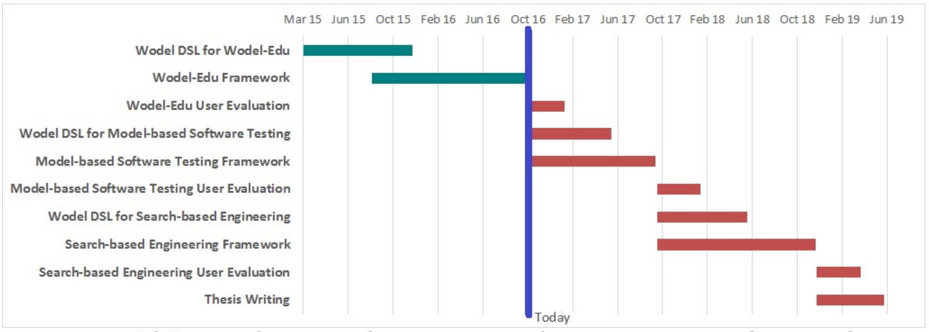

Fig. 2: PhD. timeline. Total time is 1551 days $\simeq$ 4 years and 3 months

References

1. Aranega, V., Mottu, J.M., Etien, A., Degueule, T., Baudry, B., Dekeyser, J.L.: Towards an automation of the mutation analysis dedicated to model transformation. Softw. Test. Verif. Reliab. 25(5-7), 653–683 (Aug 2015)
2. Bartel, A., Baudry, B., Munoz, F., Klein, J., Mouelhi, T., Traon, Y.L.: Model driven mutation applied to adaptative systems testing. In: Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on. pp. 408–413 (March 2011)
3. Bombieri, N., Fummi, F., Guarnieri, V., Pravadelli, G.: Testbench qualification of systemc tlm protocols through mutation analysis. IEEE Trans. Comput. (2014)

[2] `http://aida.ii.uam.es/`

4. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool, USA (2012)
5. Cuadrado, J.S., Guerra, E., d. Lara, J.: Uncovering errors in atl model transformations using static analysis and constraint solving. In: 2014 IEEE 25th International Symposium on Software Reliability Engineering. pp. 34–44 (Nov 2014)
6. Deng, L., Offutt, J., Ammann, P., Mirzaei, N.: Mutation operators for testing android apps. Information and Software Technology (2016)
7. Fleck, M., Troya, J., Wimmer, M.: Search-Based Model Transformations with MOMoT, pp. 79–87. Springer International Publishing, Cham (2016)
8. Gómez-Abajo, P., Guerra, E., de Lara, J.: Wodel: A domain-specific language for model mutation. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. pp. 1968–1973. SAC '16, ACM, New York, NY, USA (2016)
9. Granda, M.F., Condori-Fernández, N., Vos, T.E.J., Pastor, O.: Mutation Operators for UML Class Diagrams, pp. 325–341. Springer International Publishing (2016)
10. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Comput. Surv. 45(1) (Dec 2012)
11. Henard, C., Papadakis, M., Traon, Y.L.: Mutalog: A tool for mutating logic formulas. In: Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on. pp. 399–404 (March 2014)
12. Krall, J., Menzies, T., Davies, M.: Gale: Geometric active learning for search-based software engineering. IEEE Transactions on Software Engineering (2015)
13. Lackner, H., Schmidt, M.: Towards the assessment of software product line tests: A mutation system for variable systems. In: Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2. pp. 62–69. SPLC '14, ACM, New York (2014)
14. Mens, T., Van Gorp, P.: A taxonomy of model transformation. Electron. Notes Theor. Comput. Sci. 152, 125–142 (Mar 2006)
15. Moawad, A., Hartmann, T., Fouquet, F., Nain, G., Klein, J., Bourcier, J.: Polymer: A model-driven approach for simpler, safer, and evolutive multi-objective optimization development. In: Model-Driven Engineering and Software Development (MODELSWARD), 2015 3rd International Conference on. pp. 1–8 (Feb 2015)
16. Offutt, A.J., Lee, A., Rothermel, G., Untch, R.H., Zapf, C.: An experimental determination of sufficient mutant operators. ACM TSEM 5(2), 99–118 (1996)
17. Pietsch, P., Yazdi, H.S., Kelter, U.: Controlled generation of models with defined properties. In: Software Engineering 2012: Fachtagung des GI-Fachbereichs Softwaretechnik, 27. Februar - 2. März 2012 in Berlin. pp. 95–106 (2012)
18. Sadigh, D., Seshia, S.A., Gupta, M.: Automating exercise generation: A step towards meeting the mooc challenge for embedded systems. In: Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education. pp. 2:1–2:8. WESE '12, ACM, New York, NY, USA (2013)
19. da Silva, A.R.: Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures 43, 139 – 155 (2015)
20. da Silva Simão, A., Maldonado, J.C.: Mudel: a language and a system for describing and generating mutants. J. Braz. Comp. Soc. 8(1), 73–86 (2002)
21. Stephan, M., Alalfi, M.H., Stevenson, A., Cordy, J.R.: Using mutation analysis for a model-clone detector comparison framework. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 1261–1264. ICSE '13, IEEE Press, Piscataway, NJ, USA (2013)
22. Vincenzi, A.M.R., Simão, A.S., Delamaro, M.E., Maldonado, J.C.: Muta-pro: Towards the definition of a mutation testing process. Journal of the Brazilian Computer Society 12(2), 49–61 (2006)