Mutation Testing for DSLs (Tool Demo)

Pablo Gómez-Abajo
pablo.gomeza@uam.es
Universidad Autónoma de Madrid
Spain

Esther Guerra
esther.guerra@uam.es
Universidad Autónoma de Madrid
Spain

Juan de Lara
juan.delara@uam.es
Universidad Autónoma de Madrid
Spain

Mercedes G. Merayo
mgmerayo@fdi.ucm.es
Universidad Complutense de Madrid
Spain

Abstract

Mutation testing (MT) is a well-known technique to evaluate and improve the quality of a given test-suite. While several MT tools exist for traditional programming languages, there is no systematic method to create MT tools for domain-specific languages (DSLs). To improve this situation, we present Wodel-Test, a domain-independent tool to synthesize MT tools for arbitrary DSLs.

CCS Concepts • **Software and its engineering** → **Model-driven software engineering**; **Domain specific languages**.

Keywords Domain-specific languages, mutation testing

ACM Reference Format:
Pablo Gómez-Abajo, Esther Guerra, Juan de Lara, and Mercedes G. Merayo. 2019. Mutation Testing for DSLs (Tool Demo). In *Proceedings of the 17th ACM SIGPLAN International Workshop on Domain-Specific Modeling (DSM '19), October 20, 2019, Athens, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3358501.3361234

1 Introduction

Mutation testing (MT) is used to evaluate and improve the quality of a test-suite for a given program. It involves injecting artificial faults on the program to create *mutants*, which are executed against the test-suite. When the fault is detected, the mutant is *killed*, otherwise, it is *alive*. Equivalent mutants are live mutants that cannot be distinguished from the original program, so no test case can kill them. The ratio of artificial faults detected by the test-suite over the total number of non-equivalent mutants gives the mutation score [7, 12]. The lower the mutation score, the worse the quality of the test suite, and more test cases should be added.

Domain-specific languages (DSLs) are increasingly used, e.g., in process modelling [2], web applications [5], or distributed reactive systems [13]. While MT is commonly used for traditional programming languages like Java [4, 17, 19], C [1, 16], or C++ [6, 18], creating a MT tool is costly and error-prone, mainly due to the need to parse and modify the programs of the target language, and to define the mutation operators. As MT tools are normally programmed by hand, their creation is not "cost-effective" for their usage over DSLs, usually with a smaller base of users and fewer developers (e.g., see the MT tools for logic formulae [14] or adaptive systems [3]). Due to the high variety of DSLs, we propose Wodel-Test [11], a domain-independent framework to generate MT tools which can help to solve these problems.

DSM '19, October 20, 2019, Athens, Greece
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6984-8/19/10...$15.00
https://doi.org/10.1145/3358501.3361234

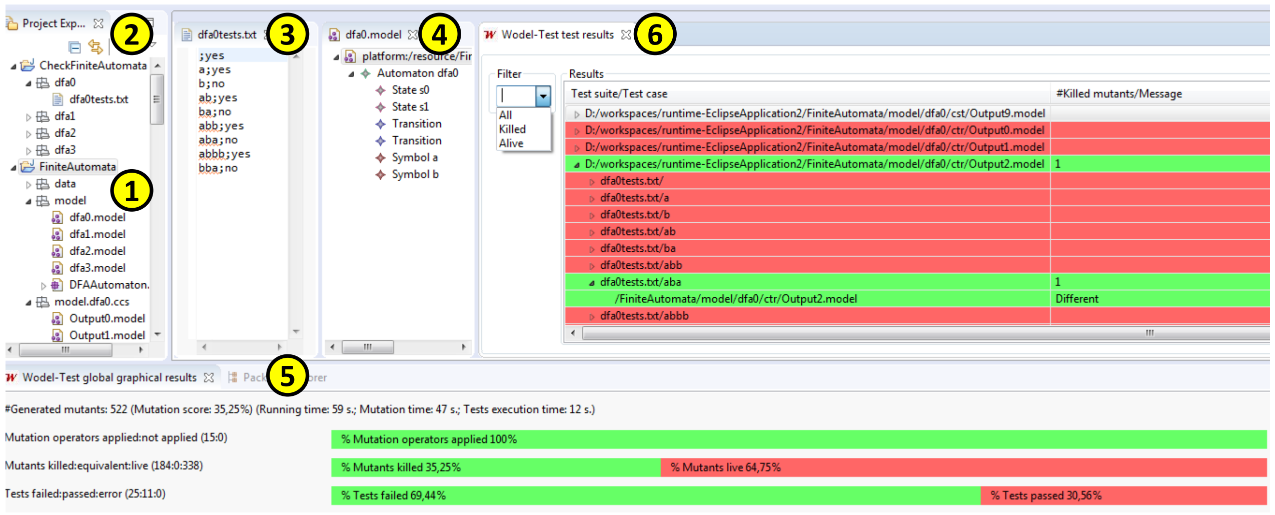
2 Defining MT Tools with Wodel-Test

Wodel-Test relies on the DSL Wodel to define model mutations for the domain meta-model [9]. Wodel has mutation primitives to *create*, *clone*, *select*, *modify*, *retype* and *delete* model elements. Listing 1 shows a mutation operator implemented in Wodel for the Finite Automata (FA) domain. Line 2 states the domain meta-model "http://fa.com", externally defined with Ecore [21]. The rts mutation operator creates a final State named f in line 5, and modifies the target of a random Transition in line 6 to point to the State created in line 5. The exhaustive mode indicated in line 1 generates all possible mutants for the given mutation operators. Alternatively, the user can set the number of mutants to generate and the process is stochastic. The Wodel program is applied over a set of seed models conformant to the given meta-model.

```
1  generate exhaustive mutants in "out/" from "models/"
2  metamodel "http://fa.com"
3  with blocks {
4    rts {
5      s = create State with {name = 'f', isFinal = true}
6      modify target tar from one Transition to s } }
```

Listing 1. A Wodel mutation operator for FA.

Wodel defines an extension point for specific post-processing applications [10]. We have used this extension point to implement a domain-independent application called Wodel-Test



CheckFiniteAutomata
- dfa0
  - dfa0tests.txt
- dfa1
- dfa2
- dfa3

FiniteAutomata
- data
- model
  - dfa0.model
  - dfa1.model
  - dfa2.model
  - dfa3.model
  - DFAAutomaton.
- model.dfa0.ccs
  - Output0.model
  - Output1.model

```
;yes
a;yes
b;no
ab;yes
ba;no
abb;yes
aba;no
abbb;yes
bba;no
```

Test suite/Test case	#Killed mutants/Message
D:/workspaces/runtime-EclipseApplication2/FiniteAutomata/model/dfa0/cst/Output9.model	
D:/workspaces/runtime-EclipseApplication2/FiniteAutomata/model/dfa0/ctr/Output0.model	
D:/workspaces/runtime-EclipseApplication2/FiniteAutomata/model/dfa0/ctr/Output1.model	
D:/workspaces/runtime-EclipseApplication2/FiniteAutomata/model/dfa0/ctr/Output2.model	1
dfa0tests.txt/	
dfa0tests.txt/a	
dfa0tests.txt/b	
dfa0tests.txt/ab	
dfa0tests.txt/ba	
dfa0tests.txt/abb	
dfa0tests.txt/aba	1
/FiniteAutomata/model/dfa0/ctr/Output2.model	Different
dfa0tests.txt/abbb	

#Generated mutants: 522 (Mutation score: 35,25%) (Running time: 59 s.; Mutation time: 47 s.; Tests execution time: 12 s.)

Mutation operators applied:not applied (15:0)	% Mutation operators applied 100%	
Mutants killed:equivalent:live (184:0:338)	% Mutants killed 35,25%	% Mutants live 64,75%
Tests failed:passed:error (25:11:0)	% Tests failed 69,44%	% Tests passed 30,56%

Figure 1. MT tool for Finite Automata generated with WODEL-TEST.

that generates MT tools for arbitrary DSLs. The process involves two user roles: the *MT tool creator* and the *tester*. The former provides the MT tool specification, from which a MT tool is generated. The latter uses the generated MT tool to perform the MT process on test-suites of DSL programs.

The first step is the MT tool creation. For this purpose, WODEL-TEST provides an extension point where the MT tool creator has to include the following three specifications:

- *language support*: This includes the DSL meta-model, a transformation to parse the textual representation of programs into models, and a transformation to serialize the models into text. For our running example, these transformations are not needed, as we directly work with models.
- *mutation support*: This consists of the definition of mutation operators for the DSL in WODEL. For our example, we have included the 15 mutation operators devised in [8, 20]. The WODEL tool computes metrics for mutation footprints which show the coverage of the DSL meta-model [10], hence helping to identify omissions and lack of coverage. In addition, WODEL-TEST provides an extension point to customize synctactic and semantic equivalence criteria for the specific DSL. The synctactic equivalence criteria detect if the extracted model-based representation of two programs have the same objects and relations. The semantic equivalence criteria detect whether two artifacts behave in the same way, e.g., two automata that accept the same language are semantically equivalent. Our example includes the FA equivalence detection algorithm [15].
- *execution support*: This is the code needed to compile the programs of the DSL and run the test suite.

WODEL-TEST takes this specification as input and automatically synthesizes a MT tool for the DSL. The tester can use the generated MT tool providing the program under test and a set of test cases. Once executed the MT process, the MT tool yields some reports which include the mutation score.

3 Generated MT Tool

Fig. 1 shows the MT tool generated for FA. First, the tester selects the FA models under test (label 1). Next, he/she provides a test-suite for them, which consists of a set of plain text files with pair values: an input string and a flag indicating whether the FA accepts the string (label 2). Label 3 shows one test case for the FA named dfa0, and label 4 shows one seed FA model. Then, the tester can select the mutation operators to apply, among the ones specified by the MT tool creator. As a result of the MT process, the tool shows the global results (e.g., a mutation score of 35.25%) in label 5. The view with label 6 summarizes the mutants detected by each test case, i.e., if the result of processing the test input string by the mutant FA is different to the original result, then the mutant is killed, otherwise the mutant is alive. The view uses the colors red/green to indicate whether a test passed or failed, and enables their filtering. Another view not shown in Fig. 1 lists the generated mutants by each mutation operator.

We have used WODEL-TEST to build MT tools for Java and ATL [11]. Videos illustrating their use are available at https://youtu.be/jbLxW2AOY3A. The specification of the MT tool for FA has 144 LOC (305 LOC for Java, 327 LOC for ATL), from which we obtained fully-functional MT tools.

4 Conclusions and Future Work

This paper presented WODEL-TEST, a tool to build MT tools for arbitrary DSLs. We reckon our tool can ease the creation of MT tools for the MDE community. In the future, we plan to create MT tools for other DSLs.

Acknowledgments

Work funded by the Spanish Ministry of Science (RTI2018-095255-B-I00) and the Madrid Region (S2018/TCS-4314).

References

[1] Hiralal Agrawal, Richard A. DeMillo, Bob Hathaway, William Hsu, Wynne Hsu, E.W. Krauser, R.J. Martin, Aditya P. Mathur, and Eugene Spafford. 1989. *Design of Mutant Operators for the C Programming Language*. Technical Report. Purdue University.

[2] Thomas Allweyer. 2010. *BPMN 2.0*. BoD.

[3] Alexandre Bartel, Benoit Baudry, Freddy Munoz, Jacques Klein, Tejeddine Mouelhi, and Yves Le Traon. 2011. Model Driven Mutation Applied to Adaptative Systems Testing. In *Proc. Mutation Analysis Workshop*. 408–413.

[4] Jeremy S. Bradbury, James R. Cordy, and Juergen Dingel. 2006. Mutation Operators for Concurrent Java (J2SE 5.0). In *2nd Workshop on Mutation Analysis (Mutation 2006 - ISSRE Workshops 2006)*. 83–92. https://doi.org/10.1109/MUTATION.2006.10

[5] Stefano Ceri, Piero Fraternali, and Aldo Bongio. 2000. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks* 33, 1 (2000), 137 – 157. https://doi.org/10.1016/S1389-1286(00)00040-2

[6] Pedro Delgado-Pérez, Inmaculada Medina-Bulo, Francisco Palomo-Lozano, Antonio García-Domínguez, and Juan José Domínguez-Jiménez. 2017. Assessment of class mutation operators for C++ with the MuCPP mutation system. *Information & Software Technology* 81 (2017), 169–184. https://doi.org/10.1016/j.infsof.2016.07.002

[7] Alex Denisov and Stanislav Pankevich. 2018. Mull It Over: Mutation Testing Based on LLVM. In *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 25–31.

[8] Pablo Gómez-Abajo, Esther Guerra, and Juan de Lara. 2016. Wodel: a domain-specific language for model mutation. In *Proceedings of the 31st ACM/SIGAPP Symposium on Applied Computing, SAC*. ACM, 1968–1973. https://doi.org/10.1145/2851613.2851751

[9] Pablo Gómez-Abajo, Esther Guerra, and Juan de Lara. 2017. A domain-specific language for model mutation and its application to the automated generation of exercises. *Computer Languages, Systems & Structures* 49 (2017), 152 – 173.

[10] Pablo Gómez-Abajo, Esther Guerra, Juan de Lara, and Mercedes G. Merayo. 2018. A tool for domain-independent model mutation. *Science of Computer Programming* 163 (2018), 85–92.

[11] Pablo Gómez-Abajo, Esther Guerra, Juan de Lara, and Mercedes G. Merayo. 2018. Towards a model-driven engineering solution for language independent mutation testing. In *Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*. Biblioteca digital SISTEDES, 4pps.

[12] Alex Groce, Josie Holmes, Darko Marinov, August Shi, and Lingming Zhang. 2018. An Extensible, Regular-expression-based Tool for Multi-language Mutant Generation. In *International Conference on Software Engineering (ICSE)*. ACM, New York, NY, USA, 25–28.

[13] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. 2016. ThingML: a language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, October 2-7, 2016*. 125–135. http://dl.acm.org/citation.cfm?id=2976812

[14] Christopher Henard, Mike Papadakis, and Yves Le Traon. 2014. MutaLog: A Tool for Mutating Logic Formulas. In *Proc. International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE CS, 399–404.

[15] John E. Hopcroft and Richard M. Karp. 1971. *A Linear Algorithm for Testing Equivalence of Finite Automata*. Technical Report 0. Dept. of Computer Science, Cornell U.

[16] Yue Jia and Mark Harman. 2008. MILU: A Customizable, Runtime-Optimized Higher Order Mutation Testing Tool for the Full C Language. In *Testing: Academic Industrial Conference - Practice and Research Techniques (taic part 2008)*. 94–98. https://doi.org/10.1109/TAIC-PART.2008.18

[17] Sun-Woo Kim, John A. Clark, and John A. McDermid. 2001. Investigating the effectiveness of object-oriented testing strategies using the mutation method. *Softw. Test., Verif. Reliab.* 11, 3 (2001), 207–225. https://doi.org/10.1002/stvr.238

[18] Markus Kusano and Chao Wang. 2013. CCmutator: A Mutation Generator for Concurrency Constructs in Multithreaded C/C++ Applications. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 722–725. https://doi.org/10.1109/ASE.2013.6693142

[19] Yu-Seung Ma, Yong Rae Kwon, and Jeff Offutt. 2002. Inter-Class Mutation Operators for Java. In *13th International Symposium on Software Reliability Engineering (ISSRE)*. 352–366. https://doi.org/10.1109/ISSRE.2002.1173287

[20] Dorsa Sadigh, Sanjit A. Seshia, and Mona Gupta. 2013. Automating Exercise Generation: A Step Towards Meeting the MOOC Challenge for Embedded Systems. In *WESE*. ACM, Article 2, 8 pages.

[21] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: eclipse modeling framework*. Pearson Education.