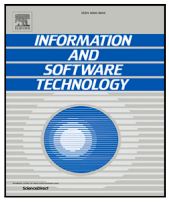
Information and Software Technology 157 (2023) 107164

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Automated engineering of domain-specific metamorphic testing environments

Pablo Gómez-Abajo [a], Pablo C. Cañizares [a], Alberto Núñez [b], Esther Guerra [a], Juan de Lara [a,*]

[a] Modelling & Software Engineering Research Group, Universidad Autónoma de Madrid, Spain
[b] Formal Analysis and Design of Software Systems Research Group, Universidad Complutense de Madrid, Spain

ARTICLE INFO

Keywords:
Metamorphic testing
Model-driven engineering
Domain-specific languages
Cloud computing
Simulation

ABSTRACT

Context: Testing is essential to improve the correctness of software systems. Metamorphic testing (MT) is an approach especially suited when the system under test lacks oracles, or they are expensive to compute. However, building an MT environment for a particular domain (e.g., cloud simulation, model transformation, machine learning) requires substantial effort.
Objective: Our goal is to facilitate the construction of MT environments for specific domains.
Method: We propose a model-driven engineering approach to automate the construction of MT environments. Starting from a meta-model capturing the domain concepts, and a description of the domain execution environment, our approach produces an MT environment featuring comprehensive support for the MT process. This includes the definition of domain-specific metamorphic relations, their evaluation, detailed reporting of the testing results, and the automated search-based generation of follow-up test cases.
Results: Our method is supported by an extensible platform for Eclipse, called Gotten. We demonstrate its effectiveness by creating an MT environment for simulation-based testing of data centres and comparing with existing tools; its suitability to conduct MT processes by replicating previous experiments; and its generality by building another MT environment for video streaming APIs.
Conclusion: Gotten is the first platform targeted at reducing the development effort of domain-specific MT environments. The environments created with Gotten facilitate the specification of metamorphic relations, their evaluation, and the generation of new test cases.

1. Introduction

Testing is a common technique for software quality assurance, used to determine whether a program behaves as expected. While there are different levels of testing (unit, integration, system, acceptance), they typically require creating *test cases* maximising the chances of finding errors (since exhaustive testing is normally not possible), together with *oracles* that predict the expected output when the system under test (SuT) is exercised with the test cases.

Metamorphic testing (MT) [1,2] is a technique for testing systems in cases where there is no oracle or it is too expensive to compute [3]. This kind of testing exploits relations (called metamorphic relations, MRs) that describe expected variations in the output of two subsequent test cases, when the input of the first test case is changed according to some criteria. For example, consider the operation $sp(G, s, d)$, returning the shortest path from a source vertex s to a destination vertex d in a graph G. If swapping the source and destination vertices, the length of the shortest path should be equal. This can be expressed as the MR $|sp(G, s, d)| = |sp(G, d, s)|$. This relation serves as a partial oracle (i.e., it tells the expected result when swapping any given source and destination vertices) and enables the generation of follow-up test cases from a given test case (i.e., the generation of the test input (G, d, s) from (G, s, d)).

MT has been successfully applied to difficult-to-test systems in many disciplines – like Web services, computer graphics, finance, or cybersecurity, among others [2] – to solve the oracle issue and automate test case generation [4,5]. However, building a dedicated MT environment for a given problem or domain requires substantial effort [6–8]. Moreover, the MRs are typically hard-coded within the MT environment [7,9–11], making it difficult to change the defined relations, to add and evaluate new relations, or to profit from the added relations to generate sensible follow-up test cases.

* Corresponding author.
E-mail addresses: Pablo.GomezA@uam.es (P. Gómez-Abajo), Pablo.Cerro@uam.es (P.C. Cañizares), Alberto.Nunez@pdi.ucm.es (A. Núñez), Esther.Guerra@uam.es (E. Guerra), Juan.deLara@uam.es (J. de Lara).

https://doi.org/10.1016/j.infsof.2023.107164
Received 2 May 2022; Received in revised form 30 December 2022; Accepted 29 January 2023
Available online 2 February 2023

Our goal is to enable the automated construction of MT environments for any domain. For this purpose, we use software language engineering [12] and model-driven engineering (MDE) [13]. MDE promotes models as the main artefacts of the software development process, and their specification by means of domain-specific languages (DSLs) [14]. Specifically, this paper contributes an MDE solution and supporting tool, called Gotten (Generic MDE framework fOr meTamorphic TEstiNg). Gotten is generic, as it enables creating MT environments for any domain that can be defined by a meta-model. The creation of an MT environment with Gotten requires providing a meta-model describing the domain concepts (i.e., the structure of the inputs of the SuT), a set of MRs, and specifications of how to execute the SuT and generate follow-up test cases. Both the MRs and the follow-up test case generation process are defined using DSLs tailored for that purpose. Starting from this information, Gotten generates an environment customised for that domain, with comprehensive support for MT, including follow-up test case generation. Hence, Gotten is generic for any domain, while each MT environment created with Gotten is specific for a domain.

Gotten is available as an Eclipse plugin at https://g0tten.github.io/gotten/. In this paper, we demonstrate its usefulness by the construction of an MT environment to test and compare simulators for data centre (i.e., cloud) architectures. Moreover, we evaluate the effectiveness of Gotten to specify MT environments, the suitability of Gotten environments to conduct MT, and its generality by creating another MT environment for video streaming APIs [15].

This paper extends [16] as follows. We have realised our framework on a tool atop Eclipse. We have expanded the design to include a DSL and a generator of follow-up test cases based on model search [17]. Finally, we have evaluated the MT environments produced by Gotten on a case study consisting of the simulation-based testing of cloud systems, compared those environments with similar MT tools developed ad-hoc, and built an MT environment for video streaming APIs.

In the following, Section 2 introduces MT and a running example in the area of simulation of data centres. Then, Section 3 overviews our approach. Section 4 introduces our DSL to specify MRs, and Section 5 our DSL to fine-tune the generation of follow-up test cases. Section 6 explains the architecture and tool support of our solution, and Section 7 reports its evaluation. Finally, Section 8 compares with related work, and Section 9 ends with the conclusions.

2. Metamorphic testing

This section provides background on MT (Section 2.1) and introduces a running example in the domain of simulation of cloud systems (Section 2.2).

2.1. Metamorphic testing (MT)

Testing complex systems entails two main challenges. The first one is the *oracle problem*, which refers to the availability of mechanisms to assess if a test case passes or fails. However, some systems – like the cloud [18], scientific software [19], or machine learning applications [20] – may not have an oracle available, or it can be computationally too expensive to apply. Additionally, since the amount of potential test cases for a complex system may be computationally unaffordable, it is desirable to select just a subset of them that assesses the system correctness effectively. Unfortunately, selecting an optimal subset is challenging in most cases. This is known as the *reliable test set problem*.

MT aims at alleviating these two problems. It has been used in very different domains, such as Web services [15], machine learning [20], compilers [21] and cloud systems [9,18]. MT uses MRs to determine if the execution of the test cases is correct. In contrast to conventional testing, where the result of each individual test case is compared to the result provided by the oracle, MT studies the relations between different test inputs and the resulting outputs.

MRs model the expected behaviour of the SuT. An MR can be seen as a property of the system involving multiple inputs and their outputs. It is represented as a logical implication $MR_i \Rightarrow MR_o$, where MR_i is a relation between the inputs of (typically two) test cases, and MR_o is a relation between their outputs. If the relation between the inputs is satisfied, so must be the relation between the outputs. As an example, given the operation $sum(a, b)$ calculating the sum of two numbers, we may define the MR: $sum(a, b) = sum(c, d) \Rightarrow sum(sum(a, x), b) = sum(sum(c, x), d)$. This relation expresses that, if the result of two sums are equal (MR_i), then the result of the same sums but increasing the first operand by an arbitrary number x should be equal (MR_o).

MT tackles the *reliable test set problem* by generating *follow-up* test cases using the MRs and a set of initial test cases, typically created by hand. A follow-up test case t' is generated from a test case t and an MR by modifying t's input so that MR_i is satisfied. For instance, given the previous MR and the test case input $(a = 1, b = 6)$, it is possible to generate follow-ups like $(c = 2, d = 5, x = 3)$ or $(c = 6, d = 1, x = 0)$, which satisfy $sum(a, b) = sum(c, d)$. Then, the MT process checks whether t and each synthesised follow-up test case also satisfy MR_o. This way, the *oracle problem* is alleviated by using the MRs as oracles.

2.2. Running example: MT for data centres

We illustrate our proposal in the domain of simulators for data centres, where the goal is to test simulators that reproduce the behaviour (expected processing time, resource utilisation) of data centres upon certain workload. The input of the simulators are models of data centres and workloads. Fig. 1 shows a meta-model to represent data centres, where a DataCentre is made of a Network and any number of Racks, and each Rack contains several Boards (as given by numBoards). Boards are connected via Switches and have computing nodes with characteristics described by NodeTypes.

Given a data centre model, it is difficult to state the precise processing time of a workload, or the energy consumed. However, it is feasible to establish rules (MRs) describing the effects on the expected time and energy consumption of, e.g., having more machines in the data centre or dealing with longer workloads. Hence, since it is difficult to establish individual oracles for a data centre simulator S, but we can define expert rules describing the expected effects of different data centre designs and workloads, we use MT. We define a test case as a pair (m_1, ω), where m_1 is a data centre model, and ω is a workload model (we omit its meta-model for simplicity), while $S_t(m_1, \omega)$ is the simulation time of data centre m_1 when processing the workload ω using simulator S. Then, we can exploit the expert knowledge about data centres to define MRs. For example, as a rule of thumb, decreasing the number of computing nodes (leaving the other components the same) may increase the processing time of a workload. This can be expressed as $NNodes(m_1) > NNodes(m_2) \Rightarrow S_t(m_1, \omega) \leq S_t(m_2, \omega)$, with NNodes being a function counting the number of nodes of a data centre. Function NNodes in the pre-condition is to be evaluated on the test cases, while S_t in the post-condition is evaluated on the simulation results. We call the functions over test cases (like NNodes) *input features*, and those on the outputs (like S_t) *output features*.

The MR can be used as an oracle (to check that decreasing nodes increases the simulation time) and to generate follow-up test cases (e.g., by decreasing the value of nodesPerBoard of an initial test model m_1). A thorough MT process considers not one but a catalogue of MRs modelling different aspects of the SuT (e.g., to assess how the simulator handles the network, bandwidth or energy consumption of the data centre model).

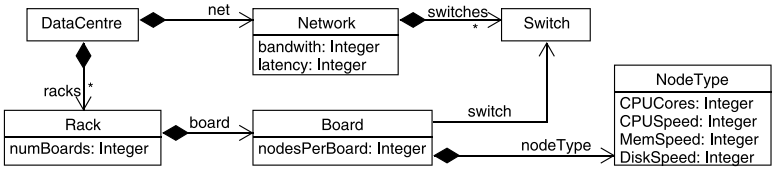


Fig. 1. Meta-model for data centres.

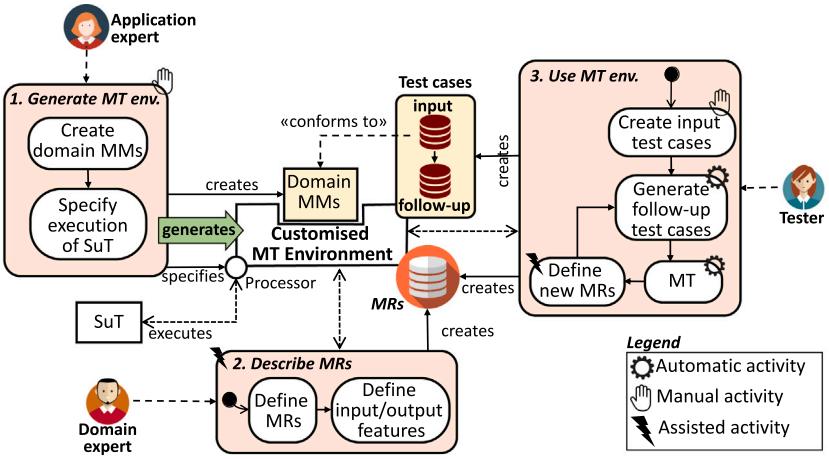


Fig. 2. Overview of our approach.

3. Overview of the approach

We propose a process to synthesise MT environments for specific domains, whose scheme is depicted in Fig. 2. It involves three roles: application expert, domain expert, and tester, who may (or may not) be the same person. The process has three phases, explained next.

(1) **Generate MT environment.** In the first phase, the application expert describes the MT environment. This expert should know the details of the SuT to be able to describe the structure of its inputs via a meta-model, as well as how to execute the SuT programmatically and obtain its results. The execution details of the SuT are provided by means of an extension point called Processor. This requires implementing three methods of a Java interface encoding how to parse the test cases into the format expected by the SuT, execute the SuT, and process its output. The result of this phase is an MT environment specific to the SuT. For our running example, the expert would need to map the data centre meta-model into the input of a specific simulator, like CloudSim [22], and would specify how to run the simulator programmatically, and how to extract the simulation outputs of interest. The resulting MT environment can be used for several SuTs, if they share the same meta-model for their inputs. A different implementation of the extension point would be needed for each SuT though. For the running example, this enables incorporating several simulators (e.g., CloudSimStorage [23], Dissect [24]) and performing their comparison using MT. Section 6 provides more details about this phase.

(2) **Describe MRs.** Next, the domain expert configures the MT environment by defining MRs modelling the behaviour of the SuT. For this purpose, we provide a dedicated DSL to define MRs, together with the input features (e.g., NNodes) and output features (e.g., S_t) that the MRs can use. The value of the input features is extracted from the test models using OCL [25]. The value of the output features is extracted from the SuT execution results by running the code provided in the previous phase for the extension point Processor. Section 4 details this DSL.

(3) **Use MT environment for testing.** Finally, the MT environment is ready for testing. First, the tester should provide manually created test models conformant to the domain meta-model provided by the application expert in phase 1. Then, the environment can be used for the following tasks:

- *Generate follow-up test cases*. Since our approach is model-based, we use model-based search engineering [17] to automatically generate follow-ups. Specifically, we provide a DSL to specify allowed changes to the input models, and then use search-based techniques to find valid follow-up test cases by iteratively applying those changes to the input models. To facilitate this task, we use heuristics to automatically synthesise these specifications from the MRs, but the tester can fine-tune the produced specifications. Section 5 explains this process in more detail.
- *MT*. The environment executes each input test model and the generated follow-ups. Then, it uses the MRs as the oracle function: when a pair of models satisfies the pre-condition of an MR, the post-condition of the MR is evaluated to determine the success or failure of the test case. When the testing process finishes, the environment reports the passed/failed test cases, the reasons for failure, and permits inspecting the input models corresponding to each test case.
- *Define new MRs*. To complement the MRs defined in phase 2, the tester can define new ones by means of the DSL for MRs.

Our solution is especially suited to build MT environments to test DSLs defined by a meta-model. In such a case, creating a domain meta-model in the first phase would not be necessary, as one would reuse the DSL's meta-model. Additionally, our approach is also suitable to create MT environments for other types of SuTs, and we will illustrate this in the domain of data centre simulators and video streaming APIs.

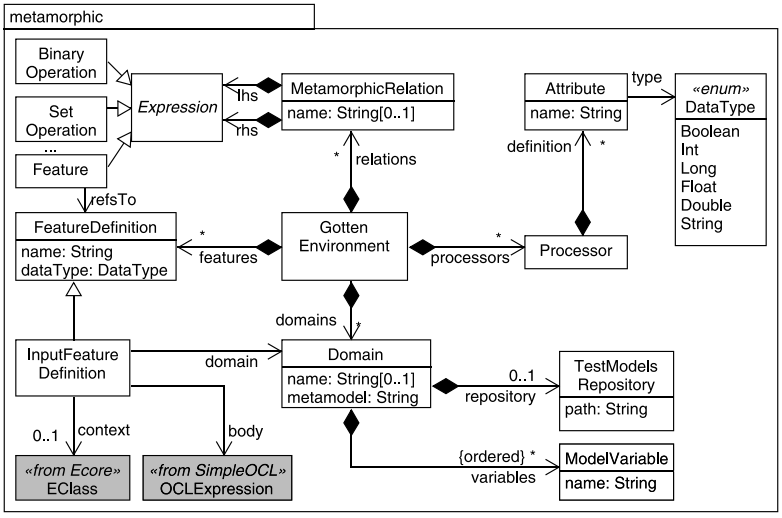


Fig. 3. Meta-model excerpt of *mrDSL*.

4. A DSL to describe metamorphic relations

Our approach offers a textual DSL to define MRs, called *mrDSL*. Fig. 3 shows an excerpt of its meta-model, whereby an MR specification (represented by class GottenEnvironment) comprises five parts:

(i) A declaration of the meta-models defining the structure of the input test cases (class Domain), the folder containing the input test models – instances of the given meta-models – for the MT process (class TestModelsRepository), and variable names representing models of the domain to be used within the MRs (class ModelVariable). For generality, we support multiple domain meta-models. For instance, our running example separates the specification of the data centre models from the workload models.

(ii, iii) A definition of the input and output features that the MRs can use (reference features). Output features (class FeatureDefinition) have a name and a type (numeric types, boolean and string). Input features (class InputFeatureDefinition), in addition, declare an OCL expression to calculate their value from the input test models. OCL [25] is a rich language based on first-order logic and set theory supporting arithmetic and set-oriented operations, quantifiers, nested predicates and conditional expressions, among others. This makes it possible to express complex input features. Input features may also be defined in the context of a class (an EClass, since we assume Ecore domain meta-models built using the Eclipse Modeling Framework [26]).

(iv) The meta-data (class Attribute) to be informed by each SuT (class Processor) that the MT process will use. In our example, this allows declaring the data that the simulators supported by the MT environment (e.g., CloudSim, Dissect) need to provide.

(v) The definition of MRs (class MetamorphicRelation) consisting of an implication, where the left and right hand sides can reference features, models and constants, and define expressions containing relational (<, >, >=, <=, ==, <>), arithmetic (+, −, ∗, /, %), logical (and, or, not) and set (includes, excludes, size) operators. Even if the current set of supported operators is limited, the input features appearing in MRs can be complex as they are expressed with OCL.

Listing 1 shows two MRs for the running example. Lines 1 and 3 declare the location of the datacentre (cf. Fig. 1) and workload meta-models. Lines 2 and 4 specify the folders containing the initial test models, which should conform to the given meta-models. For each meta-model, there are variable declarations to refer to the initial test cases (m1 and w1) and the follow-ups (m2 and w2).

```
1  metamodel datacentre "/sample.gotten/model/datac.ecore" with m1, m2
2  models "/sample.gotten/model/dcmodels"
3  metamodel workload "/sample.gotten/model/workload.ecore" with w1, w2
4  models "/sample.gotten/model/workloads"
5
6  datacentre input Features {
7      context DataCentre def: NNodes: Int =
8          racks→collect(numBoards∗board.nodesPerBoard)→sum()
9      context DataCentre def: CPU: Int =
10         racks→collect(numBoards∗board.nodesPerBoard∗
11         board.nodeType.CPUCores∗board.nodeType.CPUSpeed)→sum()
12 }
13
14 output Features {
15     Time: Long
16     Energy: Long
17 }
18
19 Processor {
20     Name: String
21     Version: String
22 }
23
24 MetamorphicRelations {
25     MR1 = [ (NNodes(m1) > NNodes(m2) and w1 == w2) implies
26             (Time(m1) <= Time(m2)) ]
27     MR2 = [ (CPU(m1) > CPU(m2) and w1 == w2) implies
28             (Energy(m1) <= Energy(m2)) ]
29 }
```

Listing 1: An *mrDSL* specification defining two MRs for the running example.

Lines 6–12 declare the input features that may appear in the MRs, which are calculated on the input test models. The listing declares the input features NNodes and CPU of the datacentre meta-model, with type Int, and the OCL expression to compute their value. The feature's context specifies the type of the object where the OCL expression is to be evaluated. Alternatively, when the OCL expression has a global scope, the context is empty. In our example, we assume that datacentre models have exactly one DataCentre object.

Lines 14–17 declare the output features Time and Energy with type Long. Since the values of the output features need to be retrieved from the SuT execution results, we provide an extension point to specify how to retrieve these values (cf. Section 6.1).

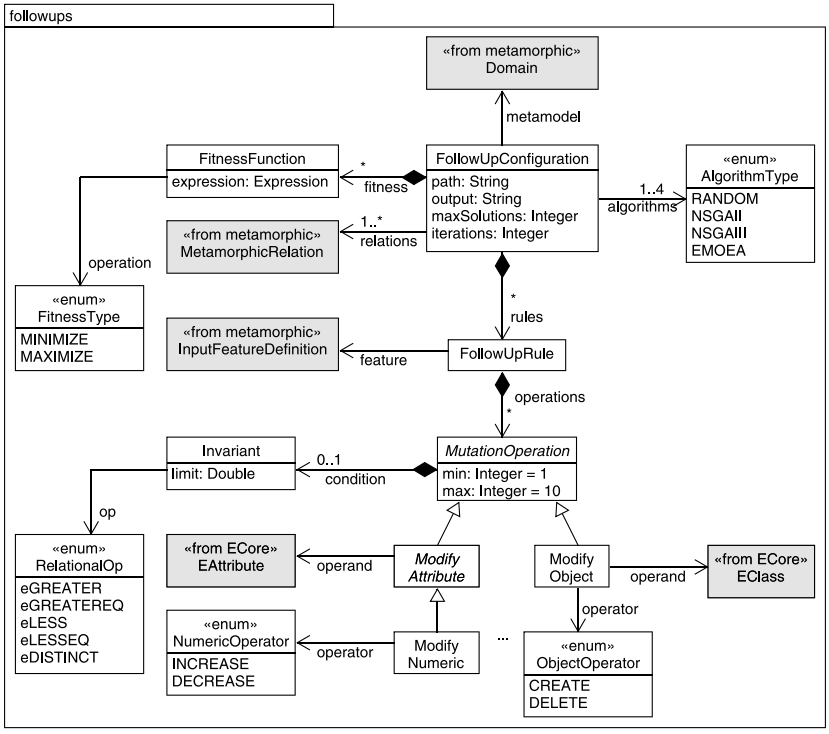


Fig. 4. Excerpt of the meta-model of *fowDSL*.

Lines 19–22 declare the processor's meta-data, in this case, Name and Version. At run-time, the generated MT environment supports instantiating multiple processors to enable the comparison of alternative SuTs. In our example, this permits comparing the accuracy of different cloud simulators w.r.t. the aspects modelled by each MR.

Finally, lines 24–29 define two MRs that use the defined input and output features. The first MR, called MR1, encodes the relation introduced in Section 2.2 (decreasing the number of nodes may increase the processing time). The second MR (MR2) states that data centres with higher computational power (CPU(m1)>CPU(m2)) require less energy consumption (Energy(m1)<=Energy(m2)) to process the same workload (w1==w2) [27].

By convention, our system silently assumes that only the features that are explicit in the MR's pre-condition (the left of the implication) change from m1 to m2. This means, e.g., that in MR1, only the NNode feature changes from m1 to m2, but other elements not considered by the MR (e.g., the Network configuration) are assumed to be the same in both models (otherwise, the MR pre-condition is not satisfied).

5. Generation of follow-up test cases

We use search-based techniques to generate follow-ups. These techniques are useful to find solutions to a given problem while optimising one or several properties [28]. This is achieved by representing the problem as a search over solution candidates, which are iteratively modified using mutation and crossover operators and evaluated using fitness criteria. In our case, the goal is finding follow-up test cases of a given one. A test case *m2* is a valid follow-up of a seed test case *m1* for a given MR, if the pre-condition of the MR is satisfied. Moreover, we assume that only the elements appearing in the pre-condition of the MR change from *m1* to *m2*. This is a convenient shortcut to avoid the need to specify the parts that remain equal in the seed and follow-up models. It makes the MR shorter and more readable, enabling better control of how the follow-ups are generated. In our example, only the elements within NNodes (in MR1) and CPU (in MR2) are allowed to change in the follow-ups. Without the assumption, we would need to specify that all other parts of the models remain equal. Generally, this assumption is a way to address the *frame problem* in logics [29].

To facilitate the configuration of the follow-ups generation process, our approach provides the textual DSL *fowDSL*. This allows customising the mutations that the search algorithms will apply to the initial seed models, and other parameters required by the search. Fig. 4 shows its meta-model.

The configuration of a follow-up generation process (class FollowUpConfiguration) comprises the following information:

(i) A reference to the meta-model of the inputs (declared with *mrDSL*), the folder containing the seed models for the search, the output folder to store the generated follow-ups, and the MR(s) the generation process will create follow-ups for (reference relations).

(ii) The definition of the mutation operations used by the search (class FollowUpRule). Each follow-up rule specifies the input feature from the *mrDSL* specification to be mutated, and the mutation operations used for that purpose (class MutationOperation). The DSL provides mutation operators for objects (creation and deletion), numeric values (adding and subtracting), boolean values (switching their value) and strings (switching, appending or deleting characters). Each operator has an interval stating the minimum and maximum number of applications. Operators may also declare an invariant (class Invariant) that should hold after executing the operator. This is useful to specify lower or upper bounds on some attribute values, or on the number of objects of a given type.

(iii) The definition of fitness functions to evaluate the suitability of the found follow-up test cases. These functions maximise or minimise a value, and are used to select the best follow-ups in each iteration of the search, i.e., those that maximise or minimise fitness values.

(iv) The evolutionary algorithm(s) to be used in the search (Random, NSGAII, NSGAIII, or eMOEA), the number of iterations of the algorithm, and the maximum number of solutions to be generated. The Random algorithm selects the solutions of each

generation to be used in the next iteration randomly. NSGAII, NSGAIII and eMOEA use an optimised selection strategy based on non-dominated sorting, where the best solutions are selected according to an ascending level of non-domination. A solution x_1 dominates another solution x_2, if x_1 is no worse than x_2 for all objectives, and is strictly better in at least one objective. If several algorithms are selected, all of them are used to generate test cases up to the specified maximum number of solutions.

To simplify the usage of the DSL, the tool can automatically generate an initial *fowDSL* configuration from an *mrDSL* specification. Algorithm 1 shows how this is performed. It receives a set of MRs described using *mrDSL* (line 1). For each MR, the algorithm performs a static analysis of the input features used in the MR's pre-condition (line 4). Then, for each input feature, it collects the elements appearing in its OCL expression (line 7). Finally, it generates suitable mutation operations depending on the collected element types, that is, it creates mutation operations of type ModifyNumeric for numeric elements, operations of type ModifyObject for objects, and so on (lines 8–15). As an illustration, Algorithm 1 shows the function creating the mutation operations for numeric elements (lines 18–31). By default, given an element, its owner feature definition, and an MR, the algorithm creates one mutation operation for each operator supported (e.g., *increase* and *decrease* for numbers, see line 25). In addition, the algorithm applies the following heuristic to reduce the number of created mutation operations by producing only the most likely ones: if the feature is used with a follow-up model (e.g., *m2*) on the left of a relational expression $<$ or $\leq$, or on the right of $>$ or $\geq$, then only a *decrease* operation is produced. Lines 20–21 show this heuristic, and lines 22–23 a symmetric case. This heuristic is also applied to object elements to decide whether producing both *create* and *delete* operations, or only one of them. The generated *fowDSL* configuration can be executed as is, or the tester can fine-tune the derived mutation operations and the other parameters for the search.

Listing 2(a) shows the *fowDSL* configuration automatically generated from MR1 in our running example. Line 1 selects the meta-model of the seed models and the MRs for the follow-up test cases. Lines 2–3 define the source folder containing the seed models, and the output folder to store the generated follow-ups. Next, lines 5–9 show the follow-up rule generated automatically for the NNodes input feature. The algorithm applies the heuristics, and so, the rule declares mutation operations to delete Rack and Board objects, and to decrease the value of Rack.numBoards and Board.nodesPerBoard. The mutation operators use the default minimum and maximum application intervals (1 and 10, cf. Fig. 4). Altogether, these operations decrement the value of NNodes.

Listing 2(b) shows a possible customisation of the configuration by a tester. The tester has left two of the mutation operations (lines 5–7) reducing its application interval to [1..4]. The operators declare invariants for the modified values using the keyword keeping. These require that the values of Rack.numBoards and Board.nodesPerBoard remain positive after applying the mutation operation. In line 9, the tester has defined a fitness function maximising the difference of the value of NNodes in the seed and follow-up models. This way, the obtained follow-up will tend to have a very different number of nodes compared to the seed test case. Finally, in lines 11–13, the search is configured to generate 10 follow-ups at most, to perform 8 search iterations, and to use all available algorithms.

Fig. 5 shows an example seed model *m1*, and a follow-up *m2* generated by Listing 2(b) from *m1*. The follow-up has lower nodesPerBoard and numBoards values. These models satisfy MR1 because NNodes(m1)=112, NNodes(m2)=32, and hence, NNodes(m1)>NNodes(m2), as MR1 requires. Since the search process only mutates nodesPerBoard and numBoards, the rest of the seed model remains intact in the follow-up.

Algorithm 1 Generating a *fowDSL* configuration from an *mrDSL* specification.

```
1: function GenFowConfiguration(rels : MetamorphicRelation[*]) : FollowUpConfiguration
2:     configuration ← new FollowUpConfiguration()
3:     for all rel in rels do
4:         for all feat in getInputFeatureDefSet(rel.lhs) do
5:             rule ← new FollowUpRule()
6:             rule.feature ← feat
7:             for all elem in getElementsSet(feat.body) do
8:                 if isNumeric(elem) then
9:                     rule.operations.addAll(CreateNumericMutationOps(elem, feat, rel))
10:                else if isObject(elem) then
11:                    rule.operations.addAll(CreateObjectMutationOps(elem, feat, rel))
12:                else if isString(elem) then
13:                    rule.operations.addAll(CreateStringMutationOps(elem, feat, rel))
14:                else if isBoolean(elem) then
15:                    rule.operations.addAll(CreateBooleanMutationOps(elem, feat, rel))
16:            configuration.rules.add(rule)
17:    return configuration

18: function CreateNumericMutationOps(elem : EStructuralFeature, feat : FeatureDefinition,
                                       rel : MetamorphicRelation) : MutationOperation[*]
19:    operations ← []
20:    if feat is left operand of {<, ≤} as follow-up in rel or
          feat is right operand of {>, ≥} as follow-up in rel then
21:        operators = [NumericOperator :: DECREASE]
22:    else if feat is left operand of {>, ≥} as follow-up in rel or
              feat is right operand of {<, ≤} as follow-up in rel then
23:        operators = [NumericOperator :: INCREASE]
24:    else
25:        operators = [NumericOperator :: INCREASE, NumericOperator :: DECREASE]
26:    for all operator in operators do
27:        operation ← new ModifyNumeric()
28:        operation.operator ← operator
29:        operation.operand ← (EAttribute)elem
30:        operations.add(operation)
31:    return operations

32: function CreateObjectMutationOps(...) : MutationOperation[*] ...
33: function CreateStringMutationOps(...) : MutationOperation[*] ...
34: function CreateBooleanMutationOps(...) : MutationOperation[*] ...
```

6. Tool support

This section describes the tool we have developed to support our proposal, focusing on its architecture (Section 6.1), the follow-ups generation process (Section 6.2), and its front-end (Section 6.3).

6.1. Architecture

Gotten is an Eclipse plugin that uses the Eclipse Modeling Framework (EMF) [26] as the underlying modelling technology. Fig. 6 shows its architecture. It provides an editor (label 1 in the figure) where domain experts and testers can specify MRs as well as input and output features using *mrDSL* (cf. Section 4). A generator (label 2) analyses the MRs and derives a convenient *fowDSL* configuration for generating follow-ups (cf. Section 5). This configuration can later be fine-tuned using a dedicated editor.

The follow-up test case generation is realised using MOMoT [17], an Eclipse plugin for search-based optimisation of problems that are expressed as models. The input to the tool is a meta-model, a seed model, a program configuring the search options, and a set of graph transformation rules [30] describing the mutation operations applicable during the search. Hence, Gotten includes a MOMoT generator (label 3) that produces all these artefacts out of *fowDSL* configurations. Section 6.2 explains this generator in more detail. Next, Gotten delegates the generation of follow-ups to MOMoT (label 4). MOMoT returns all follow-up test models it finds after the search terminates. For this purpose, MOMoT internally uses search-based evolutionary algorithms that may need to produce several generations of candidate model populations, but this is transparent to Gotten, which uses MOMoT as a black box.

```
followups for datacentre using MR1
with source folder = "/dcmodels"
and output folder = "/dcmodels"

NNodes →
   delete [1..10] Rack;
   delete [1..10] Board;
   decrease [1..10] Rack.numBoards;
   decrease [1..10] Board.nodesPerBoard

maxSolutions 10
iterations 10
algorithms [Random, NSGAII, NSGAIII, eMOEA]
```

(a) Configuration generated automatically

```
followups for datacentre using MR1
with source folder = "/dcmodels"
and output folder = "/dcmodels"

NNodes →
   decrease [1..4] Rack.numBoards keeping {Rack.numBoards > 0};
   decrease [1..4] Board.nodesPerBoard keeping {Board.nodesPerBoard > 0}

maximise ( NNodes(m2) — NNodes(m1) )

maxSolutions 10
iterations 8
algorithms [Random, NSGAII, NSGAIII, eMOEA]
```

(b) Manual modification of configuration in Listing 2(a)

Listing 2: *fowDSL* configuration for MR1 in Listing 1.

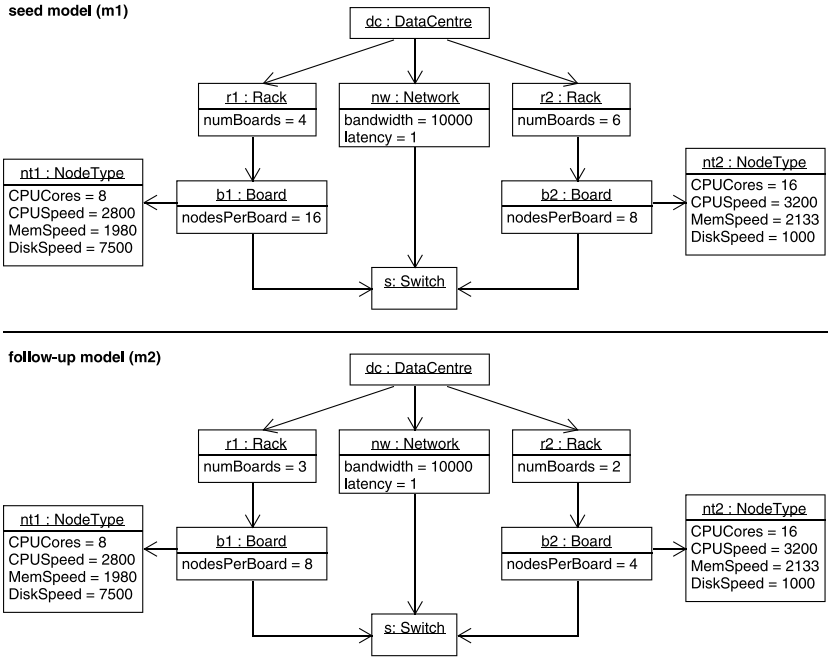


Fig. 5. A seed model and a possible follow-up model.

The last component of the architecture (label 5) is in charge of conducting the MT process. This involves executing the initial and generated follow-up test cases, using the MRs as oracles. The testing results are made available via interactive reports, which can be persisted in CSV format.

In addition, Gotten provides an extension point to gain programmatic access to the SuT. This extension point requires the implementation of an interface called Processor with three methods to transform the test case models into the input format required by the SuT (method generate), to run the SuT over the given inputs (method execute), and to obtain the value of the output features from the execution results (method getFeatures). There may be several implementations of the interface, to enable the testing and comparison of different variants of the SuT. In our running example, this allows comparing different simulators for data centres. Section 6.3 will provide more details of the front-end of Gotten.

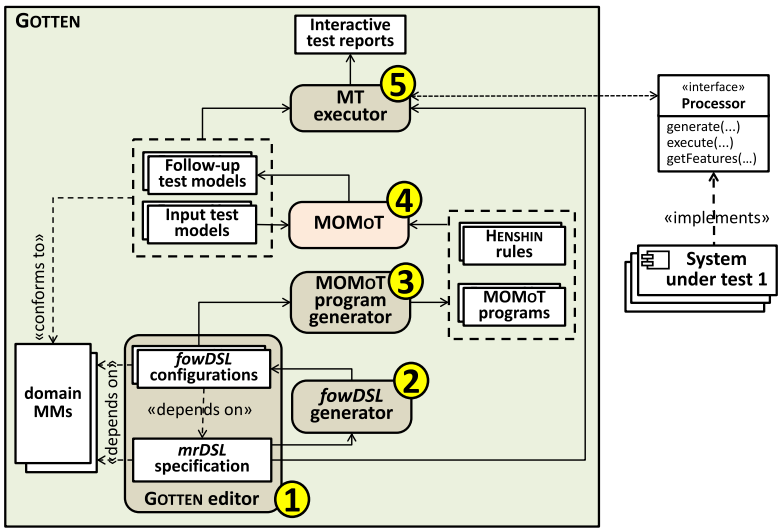


Fig. 6. Architecture of GOTTEN.

6.2. Follow-up test case generation

Given a *fowDSL* configuration, GOTTEN generates the corresponding MOMoT program [17] to perform the search of follow-up models. Since MOMoT encodes model mutation operators as graph transformation rules, these are automatically synthesised from the *fowDSL* configurations.

In an MDE setting, models (e.g., created with EMF) are encoded as graphs, so they can be manipulated by graph transformation. A graph transformation rule is made of a left- and a right-hand side graphs (LHS and RHS, respectively). If there is an occurrence of the LHS of a rule in a model, then the rule can be applied by substituting the LHS occurrence by the RHS. Technically, MOMoT uses Henshin [31] – an Eclipse plugin for graph transformation – to encode the mutation operations as transformation rules. Henshin permits structuring the rules using transformation units that implement control flow primitives, such as looping (apply a rule as long as possible), iteration (apply a rule a fixed number of times), or sequencing (apply a set of rules one after the other). Transformation units can also be nested.

The left of Fig. 7 shows a schema of the rules that GOTTEN produces from representative mutation operators in *fowDSL*. To the right, the figure shows examples of generated rules for the running example.

Fig. 7(a.1) shows the handling of the increase operator for numeric attributes. This operator increases a numeric attribute numAtt of any object of type Class, by a random number in the interval [m..M]. Accordingly, the LHS of the rule selects a suitable object ob, and the RHS increments its attribute numAtt by a random amount. Fig. 7(a.2) shows a generated rule for our running example. It uses the compact Henshin notation for rules, where the LHS and the RHS are overlapped in one graph, and elements are tagged with create (they appear on the RHS but not on the LHS), delete (they appear on the LHS but not on the RHS) or preserve (they appear on the LHS and the RHS). Attribute modifications are depicted with the notation oldValue → newValue.

Similarly, the decrease operator decrements a numeric attribute by a random amount within a given interval (cf. Figs. 7(b.1) and 7(b.2)).

The create operator creates objects of a given type. Since in EMF every object (but the root one) must have a container, the rule needs to find a compatible container for the created object (object co in Fig. 7(c.1)). Moreover, the rule is placed in an iterated unit to apply it a random number of times within the given interval. The example rule in Fig. 7(c.2) creates a Switch object within a Network, and the iterated unit repeats the creation 13 times (a random number within [1..20]).

Conversely, the delete operator removes a random number of objects of a given type (cf. Fig. 7(d.1)). It also removes the connection of the object to its container and the contents of the object (i.e., other objects accessible from it via composition references). The rule is applied using the single pushout (SPO) semantics [30], which ensures that possible links from/to the deleted objects are deleted as well. The example in Fig. 7(d.2) shows the rule resulting from the instruction delete [1..15] Board. The rule deletes a Board object, its connection to the Rack (i.e., the object container), and the NodeTypes within the object. The latter are selected using the multi-object notation, which matches all existing NodeTypes within the Board. If Board or NodeType could hold more element types via composition, these would be deleted using additional multi-objects. The iterated unit applies this rule 7 times (a number within [1..15]).

For brevity, we omit the rules to handle the mutation of boolean and String attributes.

Finally, *fowDSL* mutation operators can define invariants (keeping keyword in Listing 2(b)). These are translated into rule application conditions that need to evaluate to true for the rule to be applicable. Since the keeping invariants are tested after applying the mutation operator, they need to be translated into rule pre-conditions ensuring that, if the rule is applicable and the invariant is satisfied, the invariant still holds after the rule application. This is a restricted case of the problem of transforming an OCL post-condition into a rule pre-condition [32].

More in detail, if a decrease operator has an invariant of the form attr OP constant, with OP={>, >=}, we generate the rule pre-condition (attr - M) OP constant, with M the maximum possible random value that the rule can subtract from attr. For the <> operator we generate the pre-condition (attr - M) > constant OR (attr - m) < constant (with m the minimum possible value the rule can subtract). Similarly, if a delete operator has an invariant of the form number(Class) OP constant, which puts a bound on the number of objects of type Class, then we generate the rule pre-condition (Class::allInstances()→size() - r) OP constant, where allInstances() is the OCL operation that returns the set of instances of Class, and r the number of objects to be deleted. The rule pre-conditions generated from invariants defined in increase and create operators are generated in a similar way. Implementation-wise, the instances of a class are collected using Java, since Henshin does not support OCL.

Overall, *fowDSL* acts as a high-level interface for MOMoT, which is the tool to which GOTTEN delegates the generation of follow-ups. Using *fowDSL* shields the users from interacting directly with MOMoT, which requires a much more complex configuration, along with the specification of mutation operators in other technology (graph transformation)

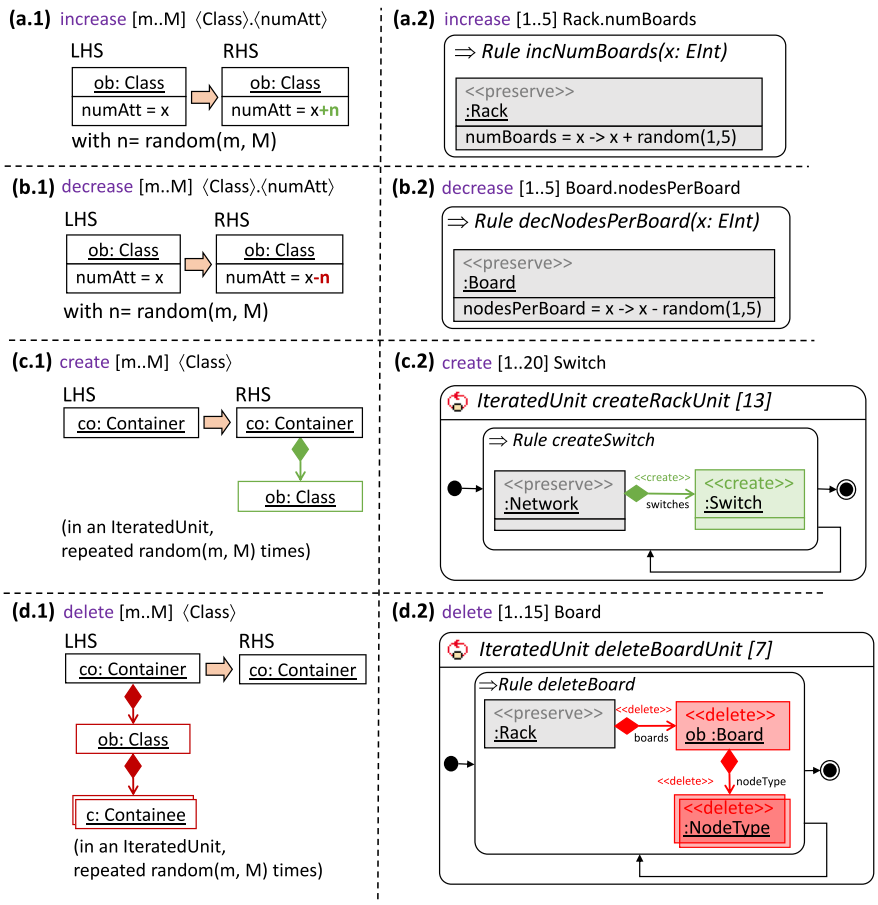


Fig. 7. Graph transformation rules generated from representative *fowDSL* mutation operations. Schema (left) and example (right).

and tool (Henshin). We opted for making the automatically generated *fowDSL* configurations explicit to the user to allow their customisation (e.g., changing or deleting mutation operators, choosing the search algorithm, etc.).

6.3. Environment

The Gotten environment is an Eclipse plugin, developed using EMF [26] for handling the (meta-)models, and Xtext [33] for creating the editor for both DSLs. More information about the tool is available at https://g0tten.github.io/gotten/, including a demonstration video.

Fig. 8 shows a screenshot of the environment. It provides editors for *mrDSL* and *fowDSL* featuring code completion, syntax highlighting and validation (labels 1 and 2 in the figure). For example, the *mrDSL* editor reports an error if an MR uses a feature not applicable to the model variables, and a warning if the right side of an MR uses an input feature.

The tool supports creating Gotten projects (label 3). These store *mrDSL* and *fowDSL* programs, and typically host the meta-models and test models. Gotten has a wizard to synthesise a *fowDSL* configuration from a given *mrDSL* specification, and another to create a plugin project template instantiating the Processor extension point. As an example, label 4 shows two projects implementing the methods of the Processor interface (cf. Fig. 6) for the execution of datacentre models using the simulators CloudSim [22] and Dissect [24].

The environment has several views for the MT process. The test execution view (label 5) classifies the test models by each active processor. This view allows to double-click on each model to launch the generate method of the instantiated Processor interface, and to execute the MT process for the selected processors by double-clicking on each of them. The view also shows the value of the output features. A wizard (not shown) permits launching the MT process for the selected processors and MRs.

The test results view (label 6) shows the results of the MT process for each MR, with one tab per available processor. The figure displays two tabs, for CloudSim and Dissect. The view reports the evaluation of the MRs for each pair of initial test/follow-up, distinguishing between those whose pre-condition (MR_i) is not satisfied (displayed in light-grey), those whose pre-condition and post-condition (MR_o) are satisfied (i.e., passing tests, in green), and those whose pre-condition holds but not the post-condition (i.e., failed tests, in red). The results can be filtered by each one of these three kinds.

The detailed test results view (label 7) provides more details about the MT execution. It shows the generated follow-ups with a comparison of the value of the features in the initial and the follow-up test models. The figure shows the values of the input feature CPU and the output feature Energy used by MR2. The results for the other MRs are shown in different tabs.

Finally, it is possible to generate a detailed CSV execution report to be analysed, e.g., in Excel. This report provides the values for each input and output feature, and the evaluation of each term used in the MRs.

7. Evaluation

Next, we evaluate Gotten to answer the following research questions (RQs):

RQ1 How effective is Gotten to specify MT environments?

RQ2 How suitable are the environments created with Gotten to conduct MT?

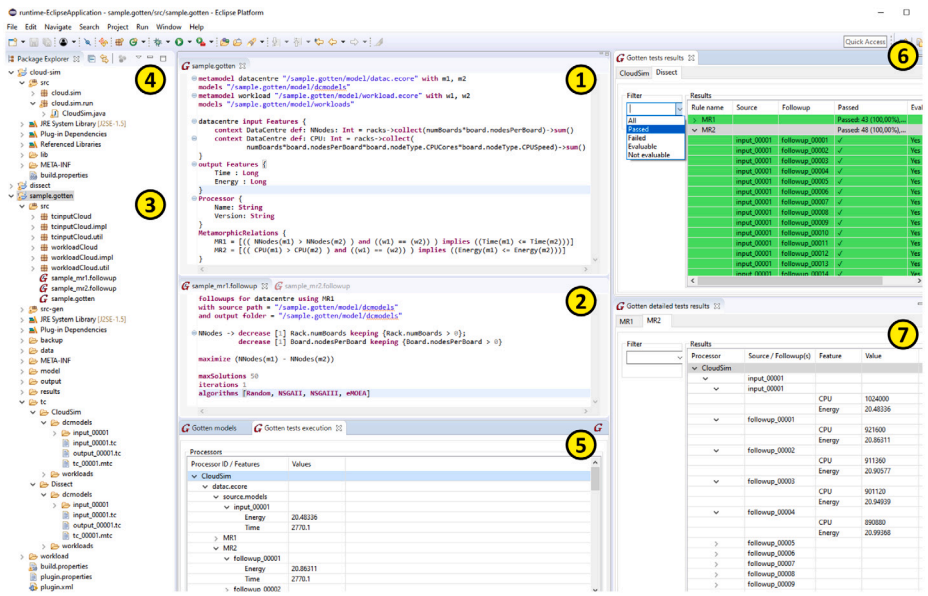


Fig. 8. GOTTEN development environment in action.

RQ3 Can GOTTEN be used to create MT environments for different domains?

We answer the first two RQs by using GOTTEN to generate an MT environment for cloud simulation. In particular, to address RQ1, Section 7.1 evaluates the cost of integrating two cloud simulators within an MT environment created with GOTTEN. Next, to respond to RQ2, Section 7.2 studies the suitability of the created environment to replicate an experiment consisting in the MT-based comparison of several cloud simulators. To answer RQ3, Section 7.3 reports on the creation of an MT tool for video streaming APIs, inspired by [15]. Finally, Section 7.4 discusses threats to validity for the evaluations.

The GOTTEN web page (https://g0tten.github.io/gotten/) contains the created MT environments and the raw data of the experiments.

7.1. RQ1: Effectiveness of GOTTEN to specify MT environments

In this section, we analyse how effective is GOTTEN to construct MT environments. For this purpose, we compare the effort required to include two processors (two cloud simulators) both in GOTTEN and in an ad-hoc MT tool called *FwCloudMeT* that some of the authors developed in previous work [27]. *FwCloudMeT* provides an ad-hoc set of MRs for cloud simulation, and its implementation of the follow-up test case generation is native (in contrast to GOTTEN, which relies on a third-party tool for that purpose). Since *FwCloudMeT* was built in Java, we compare the lines of code (LoC) of both tools. This way, LoC is used as a proxy to measure developers' effort [34].

Table 1 displays the effort required to include the Dissect [24] and CloudSimStorage [23] simulators in both environments. The first column indicates the subsystems of the tools: *Engine* is their main core, *MRs* refers to the LoC to represent the MRs, *Follow-ups* is the subsystem to generate follow-ups, and *Executor* shows the LoC required to configure and execute the simulators. In addition, the MT environment generated with GOTTEN required creating two meta-models to represent data centres (cf. Fig. 1) and workloads, which are common for both Dissect and CloudSimStorage. The data centre meta-model has 5 classes, 8 attributes and 6 references; and the workload meta-model has 2 classes, 4 attributes and 1 reference.

The LoC required to include a new processor are significantly lower in GOTTEN than in *FwCloudMeT*. Regarding the *Engine*, including each simulator into *FwCloudMeT* required more than 4300 LoC, while their inclusion in GOTTEN required less than 600 LoC. Defining the MRs and generating the follow-ups in GOTTEN only required 10 and 20 LoC, respectively, thanks to the use of our DSLs. Moreover, the MRs and follow-ups are the same for both simulators. Instead, performing these tasks in *FwCloudMeT* required around 3200 and 2500 LoC, respectively, and the code could not be reused. Finally, configuring the execution of Dissect and CloudSimStorage in GOTTEN required 564 and 509 LoC, respectively. In contrast, configuring the same simulators in *FwCloudMeT* took more than 2200 LoC.

Table 1
Effort required (in LoC) to include a new processor in an MT environment.

Subsystem	Dissect		CloudSimStorage	
	FwCloudMeT	GOTTEN	*FwCloudMeT*	GOTTEN
Engine	4341	574	4415	539
MRs	3174	10	3207	10
Follow-ups	2480	20	2513	20
Executor (processor)	2254	564	2328	509
Total	**12249**	**1168**	**12463**	**1078**

Overall, including a new processor in GOTTEN required around 1100 LoC, while including it into *FwCloudMeT* required one order of magnitude more code (around 12000 LoC). This effort may vary slightly for each processor, as each one may require additional configuration parameters for its deployment and execution. Hence, we can answer RQ1 positively: GOTTEN provides effective mechanisms to specify MT environments. Compared to creating an ad-hoc MT tool for data centre simulation, it required an order of magnitude less LoC.

7.2. RQ2: Suitability of GOTTEN environments for MT

To evaluate the suitability of our proposal, we assess if the environments created with GOTTEN are able to properly conduct the three main MT activities: defining new MRs, generating follow-ups, and performing the testing process. For this purpose, we have replicated with GOTTEN the experiments conducted in [27] using *FwCloudMeT*, an MT tool specifically built for supporting different simulation platforms. In this experiment, we integrated two of these simulators in GOTTEN: CloudSimStorage 1.0 [23] and Dissect-cf 0.9.3 [24]. Hence, we specified in GOTTEN the MRs included in [27] and the *energy* output feature, which represents the total energy consumption that a data centre requires to process a given workload. The source test cases designed in the original

Table 2
Definition of the catalogue of MRs proposed in [27] using *mrDSL*.

Relation		Description
MR1	$MR1_i$	The cloud m_1 has a better CPU system than m_2. The workloads ω_1 and ω_2 are equal.
		$MR1_i$ = [CPU(m1) > CPU(m2) **and** w1 == w2]
	$MR1_o$	The energy required to execute ω_1 over m_1 should be less than or equal to the energy required to execute ω_2 over m_2.
		$MR1_o$ = [Energy(m1) <= Energy(m2)]
MR2	$MR2_i$	The cloud m_1 contains more physical machines than the cloud m_2. The workloads ω_1 and ω_2 are equal.
		$MR2_i$ = [NMachines(m1) > NMachines(m2) **and** w1 == w2]
	$MR2_o$	The ratio between the number of machines of m_1 and m_2 should be greater than or equal to the ratio between the energy consumption required to execute ω_1 over m_1 and the one required to execute ω_2 over m_2.
		$MR2_o$ = [NMachines(m1) / NMachines(m2) >= Energy(m1) / Energy(m2)]
MR3	$MR3_i$	The cloud m_1 has a better storage system than m_2. The workloads ω_1 and ω_2 are equal.
		$MR3_i$ = [Storage(m1) > Storage(m2) **and** w1 == w2]
	$MR3_o$	The time required to execute ω_1 over m_1 should be less than or equal to the time required to execute ω_2 over m_2.
		$MR3_o$ = [Time(m1) <= Time(m2)]
MR4	$MR4_i$	The cloud m_1 has a better network system than m_2. The workloads ω_1 and ω_2 are equal.
		$MR4_i$ = [Network(m1) > Network(m2) **and** w1 == w2]
	$MR4_o$	The time required to execute ω_1 over m_1 should be less than or equal to the time required to execute ω_2 over m_2.
		$MR4_o$ = [Time(m1) <= Time(m2)]
MR5	$MR5_i$	The cloud m_1 has a better memory system than m_2. The workloads ω_1 and ω_2 are equal.
		$MR5_i$ = [Memory(m1) > Memory(m2) **and** w1 == w2]
	$MR5_o$	The time required to execute ω_1 over m_1 should be less than or equal to the time required to execute ω_2 over m_2.
		$MR5_o$ = [Time(m1) <= Time(m2)]
MR6	$MR6_i$	The clouds m_1 and m_2 are equal. The workload ω_1 contains ω_2.
		$MR6_i$ = [m1 == m2 **and** Workload(w1)—>includes(Workload(w2))]
	$MR6_o$	The time required to execute ω_2 over m_2 should be less than or equal to the time required to execute ω_1 over m_1.
		$MR6_o$ = [Time(m2) <= Time(m1)]

work were translated into EMF models, and the follow-ups were re-generated using Gotten. Next, we discuss the results for each of the three MT activities.

Defining new MRs. Table 2 shows the encoding of the MRs defined in [27] using *mrDSL*. For each MR, we show its left- and right hand sides, and a description in natural language. The DSL was expressive enough to specify all MRs, yielding a concise, intensional encoding of the relations, each one fitting in a single LoC. Please note that $MR6_i$ uses the input feature Workload to obtain the set of traces in w_1 and w_2, and the includes operator to check for inclusion.

Generating follow-ups. We were able to generate follow-ups for all the MRs using *fowDSL*. We modified the automatically generated *fowDSL* configurations to avoid numeric attributes with negative values, and to fine-tune the values in each decrease and delete mutation operation. Overall, Gotten generated 200 follow-ups for the 6 MRs. All of them satisfied the left-hand side of the MR, which means the generation process was correct. The overall generation time was 36 min and 16 s (performed on a Windows 11 PC with an Intel Core i7-10510U processor and 24 GB of RAM), which gives a generation time of less than 11 s per follow-up.

Testing process. We used the MT environment built with Gotten to conduct the testing process. Table 3 compares the results obtained with *FwCloudMeT* [27] and those obtained with Gotten.

Both tools agree that the two simulators satisfy MR1, MR3, MR4, and MR6, but not MR5. On the contrary, the percentage of test cases that satisfy MR2 differs. While 40% and 51% of the test cases satisfy MR2 in *FwCloudMeT* when using CloudSimStorage and Dissect-cf, respectively, we obtain that 45% and 80% of the test cases satisfy the same MR in Gotten. The reason for this discrepancy is two-fold. First, the random nature of the test generation algorithms used: the one of Gotten is based on search, while those of [27] use fixed heuristics. Second, the relation between the ratios of the physical machines and energy consumption is not met in the simulated environments as we expected. After careful analysis, we noticed that those scenarios providing a slightly over-dimensioned data centre keep a reduced number of physical machines idle and, consequently, do not fulfil this relation. Therefore, the number of tests that satisfy MR2 heavily depends on the number of follow-ups containing these scenarios.

In the case of MR5, none of the test cases satisfy this MR, neither using *FwCloudMeT* nor Gotten. We argue that it is because these simulators do not model the memory system accurately enough. The memory system is not a common feature in cloud simulators, which

Table 3
Comparison of results obtained in [27] and with Gotten. The cells show the percentage of tests passing each MR.

Environment	Simulator	MR1	MR2	MR3	MR4	MR5	MR6
Original results [27]	CloudSimStorage	100	40	100	100	0	100
	Dissect-cf	100	51	100	99	0	100
Proposed framework	CloudSimStorage	100	45	100	100	0	100
	Dissect-cf	100	80	100	100	0	100

tend to focus on higher-level details such as resource management policies, users, and virtualisation.

With respect to the simulation time, it only depends on the underlying simulator, but not on Gotten. For CloudSimStorage, it was 1 h and 11 min (the simulator took over 21 s per test case), and for Dissect-cf, it was 31 min (over 9 s per test case). Overall, we can answer RQ2 positively: we could perform a full MT process for the cloud simulation domain using a Gotten environment. This included defining MRs, generating test cases, and performing the testing process. Our results are in line with those obtained with *FwCloudMeT*, a specific tool for MT of cloud simulators.

7.3. RQ3: MT environments for different domains

We show Gotten's generality and flexibility by describing an additional MT environment for video streaming APIs, inspired by works on MT of REST APIs [15]. We could have created an MT environment targeting REST APIs generically. However, to show the power of our approach, we group APIs into domains, like video streaming, audio streaming, hotel bookings, or flight bookings, as Fig. 9(a) depicts. This way, we can define a single meta-model describing the domain, a catalogue of MRs for that domain, and different processors that implement the API, such as YouTube and Vimeo in the case of video streaming. This enables reusing the MR catalogue for all processors, and compare their behaviour. The input of the processors is an instance of the provided meta-model (i.e., a test case) which is executed by sending REST requests to the corresponding video streaming platform.

Fig. 9(b) shows the meta-model for video streaming APIs. Class VideoAPITest represents a test case, which has a name (attribute testName). For simplicity, a test case is made of just one operation, which can be of three kinds: search, upload and update. Upload and update require a video as parameter, described by an *id*, and optionally a title, a description, and a list of tags. Moreover, each video may provide statistical information (class VideoStatistics) like the number of visualisations and *likes* received. The output of the execution of these operations is stored in reference results.

Once the meta-model was built, we defined the MRs describing the expected behaviour of the video platform. Listing 3 illustrates some MRs, taken from the work by Segura et al. [15], which used natural language for their description. The first line of the listing declares the location of the streaming meta-model, and defines the variables m1 and m2 to represent source and follow-up test cases.

Lines 3–8 define the input features that the MRs use. Features IsSearch and IsUpdate (lines 4–5) are queries that return *true* if the test case is a search or an update operation, respectively. Similarly, MaxResults (line 6) returns the maximum number of videos requested in a search, and SearchOrder returns the sorting criterion (line 7). Lines 10–14 declare the output features NVideos, OutputVideoId and OutputVideoTitle. Their values are extracted from the result of the test execution. NVideos is the number of videos returned by a search, and OutputVideoId and OutputVideoTitle are the *id* and title of the first returned video.

Lines 18–26 define three MRs. The first one (MR1) states that, given two searches m1 and m2, if the first one requests a maximum number of videos higher than the second one (MaxResults(m1) >= MaxResults(m2)),

```
1  metamodel videostream "model/VideoStream.ecore" with m1, m2
2
3  videostream input Features {
4   context VideoAPITest def: IsSearch: Boolean=request.oclIsTypeOf(SearchVideo)
5   context VideoAPITest def: IsUpdate: Boolean=request.oclIsTypeOf(UpdateVideo)
6   context SearchVideo def: MaxResults: Int= maxResults
7   context SearchVideo def: SearchOrder: Int= orderType
8  }
9
10 output Features {
11  NVideos : Long
12  OutputVideoId: Long
13  OutputVideoTitle: String
14 }
15
16 //...
17
18 MetamorphicRelations {
19  MR1 = [ (IsSearch(m1) and MaxResults(m1) >= MaxResults(m2)) implies
20          (NVideos(m1) >= NVideos(m2))]
21  MR2 = [ (IsSearch(m1) and SearchOrder(m1) <> SearchOrder(m2)) implies
22          (NVideos(m1) == NVideos(m2))]
23  MR3 = [ IsUpdate(m1) and m1 == m2 implies
24          (OutputVideoId(m1) <> OutputVideoId(m2)) and
25          (OutputVideoTitle(m1) == OutputVideoTitle(m2)) ]
26 }
```

Listing 3: An *mrDSL* specification defining three MRs for the video streaming example.

then the actual number of videos returned by the first should be equal or higher than in the second search (NVideos(m1) >= NVideos(m2)). As explained in Section 4, Gotten assumes that a test and its follow-up only differ on the elements checked by the input features used by the MR (MaxResults in this case). For this reason, MR1 does not need to require that m2 also involves a search with the same query. Relation MR2 specifies that two searches that only differ on the ordering criterion should return the same number of videos. Finally, MR3 states that two update requests on the same video must return different identifiers but the same video title.

As the last step, we encoded two processors implementing the YouTube and Vimeo APIs. This enabled the reuse of the same meta-model, test cases and MRs to construct the MT environment for both APIs. Then, we manually designed 30 source test cases for MR1 and MR2, and automatically generated 120 follow-ups from them. The total generation time was 11 min and 56 s on a Windows 11 PC with an Intel Core i7-10510U processor and 24 GB of RAM (around 7 s per follow-up). We only generated 5 test cases for MR3 to avoid infringing the SPAM policy of YouTube. For MR1 and MR2, we designed the source test cases selecting queries based on trending searches from YouTube and Vimeo (e.g., 'world cup', 'esports', etc.) and complemented them with some corner cases extracted from Segura et al. [15] (e.g., 'winter pentathlon 1949' and 'mistrustfully'). All tests could be reused for YouTube and Vimeo. Executing all tests took 4 min and 4 s in YouTube, and 10 min and 36 s for Vimeo. For both platforms, all follow-ups for MR1 and MR3 passed. Interestingly, around 7% of tests for MR2 failed in each platform, obtaining different number of videos for search queries with different ordering criteria. The results show that the proposed framework is not only able to model the MRs provided in natural

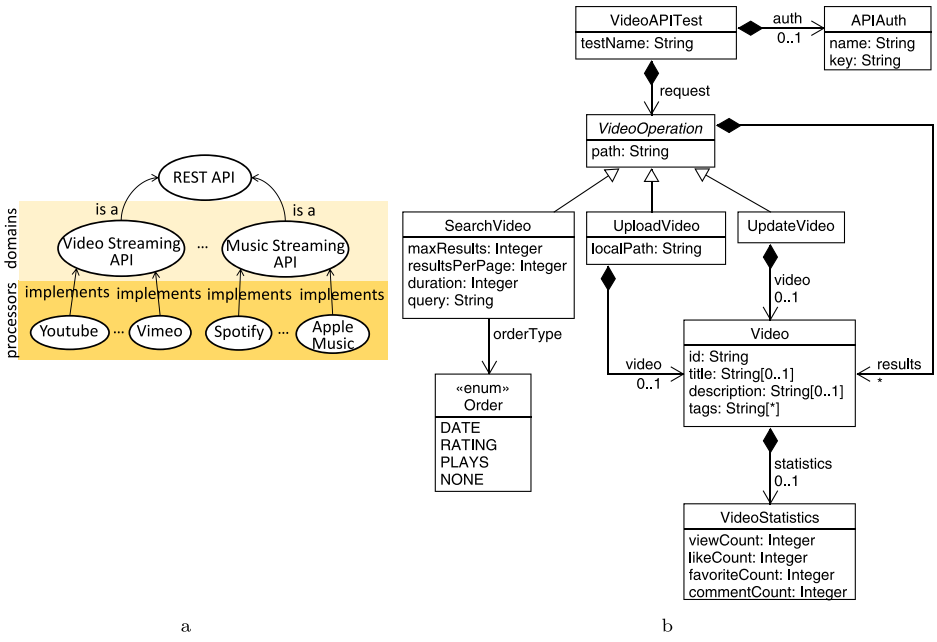


Fig. 9. (a) Organising APIs into domains. (b) Meta-model for video streaming APIs.

language by Segura et al. [15], but also to reproduce the obtained results and detect real issues[1] reported in the issue tracking systems of the studied platforms. All the artefacts involved in the definition of the MT environments, and the data of the conducted MT process, are available at https://g0tten.github.io/video/.

Overall, we can answer RQ3 positively: we could create MT environments for different domains using Gotten. By focusing on specific domains (cloud simulation, video streaming APIs) we could define an infrastructure enabling the reuse of the meta-model, the test cases, and the catalogue of MRs for different SuTs (as given by different simulators, or different video streaming APIs).

7.4. Threats to validity

Internal validity is concerned with whether our findings – based on the results of the empirical studies – truly represent a cause-and-effect relationship.

Section 7.1 compared the effort to integrate two cloud simulators into Gotten, against the effort to integrate them into *FwCloudMeT*. *FwCloudMeT* was designed and developed by one of the authors of this work, who has deep knowledge of MT, with the supervision of three experts. Gotten was designed and developed by three experts in MDE, and two of the authors performed the integration of Dissect and CloudSimStorage. Although it might be possible to optimise the code of both solutions, it should not lead to a significant alteration of the results. Consequently, we consider that the studied code bases are representative of the effort required to build an MT environment for cloud simulators.

LoC is a measure of software size often used to predict development effort [34]. We have used LoC instead of working time since we do not have an accurate measure of the hours spent in building *FwCloudMeT*.

In Section 7.2, we replicated in Gotten an experiment performed for *FwCloudMeT*. We manually and tested the correctness of the code included for the new experiment. After a careful analysis of the obtained results in Gotten and *FwCloudMeT*, we concluded that Gotten is suitable to conduct MT. However, we cannot guarantee the absence of small code errors, which the case study has not revealed.

External validity is concerned with the extent to which the results of a study can be generalised.

For RQ1 and RQ2, we mitigate external threats by using two well-known cloud simulators, widely adopted by the research community. We obtained similar results in both evaluations for both simulators. Firstly, we observe that the effort required to include each simulator is similar. Secondly, the testing results reported by Gotten and *FwCloudMeT* in the context of cloud simulation are similar, using two different simulators and six MRs. Consequently, we consider that our results can be generalised for testing cloud scenarios using simulation.

For RQ3, we used a domain (video streaming APIs) reported in the literature [15], selected to be as different as possible from the cloud simulation domain. We succeeded in building an MT environment for this new domain, and expect that the creation of MT environments for other domains be possible as long as the SuT inputs can be described by a meta-model, the testing process is not interactive, and it is feasible to feed inputs and extract outputs from the SuT programmatically.

Construct validity analyses whether the used measures are representative or not.

The study of the effort required to include a new processor into Gotten (Section 7.1) shows promising results. In the studied case, Gotten significantly reduced the LoC by an order of magnitude. Since Gotten uses DSLs to define MRs and generate follow-ups, it is possible to generate a major part of the source code for the MT process automatically. The effort to integrate each of the two simulators in Gotten was similar (see Table 1). Although including other processors in the MT environment may imply a different effort, we expect a significant effort reduction when using Gotten for this task.

Regarding the suitability evaluation, the experiments to assess the correctness of two well-known cloud simulators using *FwCloudMeT* and Gotten yield similar results (see Table 3). The differences obtained in the assessment of MR2 can be attributed to the follow-up generation strategy. Hence, we can consider that Gotten is appropriate to generate MT environments.

[1] https://code.google.com/p/gdata-issues/issues/detail?id=5173

8. Related work

Next, we review works on MT frameworks (Section 8.1), perform a feature-based comparison with Gotten (Section 8.2) and analyse works combining MT and MDE (Section 8.3).

8.1. MT frameworks

As Section 8.2 will show, existing MT frameworks are typically built ad hoc for specific domains.

Sun et al. [35,36] propose an MT framework for Web services, where MRs are described using an XML language supporting arithmetic, logical, list inclusion and relational operators. It supports the complete MT process: generating follow-ups (at random), executing and evaluating them w.r.t. MRs. In comparison, our DSL for MRs allows defining input features using OCL, an expressive constraint and navigation language. Since our MRs can become more complex than those in [35,36], we use search-based techniques for generating follow-ups.

Several approaches apply MT to autonomous driving. DeepRoad [11] is a framework for testing DNN-based autonomous driving systems, able to synthesise realistic driving scenes for detecting inconsistent driving behaviours. It uses one MR checking that the predicted car steering angle does not change under road scene transformations (e.g., from sunny to snowy weather). In the same domain, RMT [6] uses 9 MRs to add objects (e.g., pedestrians, vehicles) to existing road scenes, change the driving conditions (e.g., to night or snow), or combine both. Similarly, FogMET [37] focusses on adding fog to driving scenes.

The AMT framework [38] permits checking MRs, and generating test cases violating the MRs via constraint programming. Its scope is limited to programs supported by Inka (a subset of C and C++). Interestingly, for narrow ranges of the variables involved, the tool can prove that the SuT satisfies the MR. Instead of constraint programming, we generate follow-ups using search guided by optimisation expressions.

Hadiwijaya and Liem [39] propose a DSL for defining MRs and generating test cases for programming competitions. The DSL is defined using Xtext (like ours) and compiled into Java. It is restricted to programs with numeric inputs (e.g., optimisation problems), and MRs describe explicitly how to generate follow-ups. Instead, we support inputs with more complex structure – represented as models – and use search to generate follow-ups.

QueryFuzz [7] relies on a fixed set of MRs to test datalog engines using MT. This system has been able to detect faults in mature datalog engines.

MTKeras [10] is a generic MT tool for machine learning built atop Keras. It permits the explicit generation of follow-ups using a library of operations like adding noise or flipping an image. In the same domain, TILE [40] uses MT to check the fairness of machine learning classifiers, by means of 5 MRs that test if the classifiers treat all data in the training set equally. DeepBackground [41] uses MT to evaluate the robustness of image recognition systems based on deep learning. The tool defines 4 MRs transforming the image background. Also for image recognition, MT4ImgRec [42] proposes 6 MRs that rotate, translate or blur images. Tetraband [43] also targets image recognition machine learning models, but uses reinforcement learning to identify those MRs that are likely to reveal faults in the SuT. HydroMT [44] uses MT to assess the prediction reliability of machine learning models in the hydrological domain.

MT has also been used in the area of natural language processing. BiasFinder [45] uses MT to generate test cases to uncover bias (e.g., gender) in sentiment analysis systems. It uses bias-targeting templates (text with placeholders) to generate test data via mutation, and then checks that the sentiment labels in the test case and in the original text are the same. Similarly, MT-NLP [46] proposes analogy mutation (e.g., changing actor by actress) to discover fairness violations of language models.

MT-OSM [47] is an approach to test contributions to OpenStreetMaps. It defines 7 MRs checking elements of maps like ways (no deadlocks, at least one exit and entrance) and roundabouts (should be connected).

RISC-V [8] uses MT for processor verification. It targets the RISC-V Instruction Set Architecture, and evaluates the effectiveness of 95 MRs over faults injected on a simulator using mutation analysis. MT-EA4Cloud [9] uses MT to test simulators for cloud computing systems, based on 8 MRs about energy consumption. FwCloudMeT [27] uses MT for selecting the most appropriate simulator covering a set of features of interest using 6 MRs.

Altogether, these works evince the interest of using MT for hard-to-test programs, but they are limited to an application field. Only [10,36,39] provide dedicated languages or libraries to define MRs, but are heavily oriented to the application domain. Hence, none of them provide mechanisms to design MRs independently of the domain, execute different systems and applications, and evaluate the MRs on the results. For this purpose, our DSLs permit defining input/output features for their use in MRs, and our tooling offers extension points to link with the SuT.

8.2. Comparison of gotten and MT frameworks

Next, we report on a feature-based comparison of Gotten and the environments it generates, with the tools analysed in the previous section.

Table 4 summarises the comparison in three blocks covering different aspects of the tools. The first one (*Main Features*) shows basic information of each solution: name, creation date, year of last update, size in LoC, description of goals, implementation language and usage license. The second block (*Testing Process*) shows information related to the testing process: target domain (*Domain*), MT capabilities (*MT*), generation of follow-ups (*FU*), generation of follow-ups using search strategies (*FU Search*), size of the catalogue of MRs (*MRs*), and tool extensibility (*Extensible*). The third block (*Reporting*) shows the information that the tools provide about the testing process results. The first two columns (*UI*, *Report Type*) include the type of user interface and the way to visualise the results, while the two last columns (*Pass-Fail*, *%Pass*) report on whether the tool displays the follow-ups that pass/fail, and the percentage of passed tests.

The table analyses 20 MT solutions, including Gotten. Approximately, 56% of the analysed solutions provided updates after their release. Their implementation effort ranges from 519 to 28804 LoC. Python is the predominant implementation language, followed by Java, and there are also solutions based on R, C++, and XQuery. Nine applications do not require usage license, while the rest provides GPL, Apache 2.0, and MIT licenses.

The next block targets the MT functionality. First, the domain of the analysed applications include specific areas like autonomous driving, machine learning, image recognition and natural language processing (NLP). Instead, Gotten is the only generic tool and can be adapted to several domains.

With regards to the testing process (column *MT*), most applications use MT techniques by creating follow-ups. A few exceptions (DeepRoad, RMT, FogMet, and BiasFinder) do not support the whole MT process. Moreover, four applications (QueryFuzz, MTKeras, TILE, and RISC-V) do not generate follow-ups (column *FU*). However, these functionalities (*MT* and *FU*) are essential to automate MT, as otherwise, users must perform them manually with the subsequent effort and possibility to introduce errors. Only six tools apply search to generate quality follow-ups. Since these evaluate a specific part of the SuT, generating tests covering the most relevant parts of the SuT is crucial.

Regarding the MRs, all solutions provide a number of predefined MRs, ranging from 1 to 95. Only four solutions permit defining new MRs externally. MT4WS enables defining new MRs using XML, MTKeras relies on Python, while CheckerDSL and Gotten provide DSLs. The

Table 4
Comparison of existing MT solutions in the literature.

Main features							Testing process						Reporting			
Name	Year	Last Update	LoC	Description	Language	License	Domain	MT	FU	FU Search	MRs	Extensible	UI	Report Type	Pass-Fail	%Pass
Gotten	2022	2022	16052	Extensible platform to create MT environments for specific domains	Java	GPLv3	Generic	✓	✓	✓	✓	DSL	GUI/Eclipse	Interactive views	✓	✓
MT4WS [36]	2016	2016	Unavailable	Automated MT system for Web services	Unavailable	Private	Web services	✓	✓	✗	6	XML	Unavailable	Unavailable	✓	✓
DeepRoad [11]	2018	2018	Unavailable	GAN-based MT and input validation framework for autonomous driving systems	Unavailable	Private	Autonomous driving	✗	✓	✓	1	✗	Unavailable	Unavailable	✗	✗
RMT [6]	2020	2020	14355	Rule-based MT for autonomous driving models	Python	MIT license	Autonomous driving	✗	✓	✗	9	✗	GUI	Panel	✗	✗
FogMET [37]	2021	2021	1397	MT based on quantitative measurement for autonomous driving systems in fog	Python	GPL-3.0	Autonomous driving	✗	✓	✗	1	✗	✗	✗	✗	✗
AMT [38]	2003	2003	Unavailable	Framework to generate test cases using constraint programming	Unavailable	Private	C programs	✓	✓	✓	✓	✓	Unavailable	Unavailable	✗	✗
CheckerDSL [39]	2015	2015	3075	MT for test cases and checker generators	Java	No license	Java programs	✓	✓	Java code	10	DSL	GUI/Eclipse	Text	✗	✗
QueryFuzz [7]	2021	2022	4848	MT for datalog engines	Python	Apache 2.0	Datalog	✓	✗	✗	9	✗	Command line	✓	✗	✗
MTKeras [10]	2020	2021	1030	MT for machine learning with Keras	Python	Apache 2.0	Machine learning	✓	✗	✗	10	Python	Command line	✗	✗	✗
TILE [40]	2020	2021	836	MT of learning algorithms for balancedness	Python	No license	Machine learning	✓	✗	✗	5	✗	Command line	Text	✗	✗
DeepBackground [41]	2021	2021	2851	MT framework for deep-learning-driven images recognition accompanied with background-relevance	Python	No license	Machine learning for image recognition	✓	✓	✗	4	✗	✗	✗	✗	✗
MT4ImgRec [42]	2021	2021	2366	MT tool for image recognition software	Python	No license	Image recognition	✓	✓	✗	6	✗	GUI/Web	Panels	✗	✓
Tetraband [43]	2020	2021	1848	Adaptive MT with contextual bandits	Python	MIT license	Machine learning for image recognition	✓	✓	✓	7	✗	Command line	✓	✓	✓
HydroMT [44]	2021	2022	2444	MT for hydrological models	R	MIT license	Machine learning for hydrological models	✓	✓	✗	6	✗	Command line	Text	✗	✓
BiasFinder [45]	2021	2022	28804	MT to uncover bias in sentiment analysis systems	Python	Apache 2.0	NLP	✗	✓	✓	1	✗	Command line	✗	✗	✗
MT-NLP [46]	2020	2021	519	MT and certified mitigation of fairness violations in NLP models	Python	No license	NLP	✓	✓	✗	2	✗	Command line	Text	✗	✗
MT-OSM [47]	2020	2022	711	MT for OpenStreetMap	XQuery	No license	Open street map	✓	✓	✓	7	✗	✗	✗	✗	✗
RISC-V [8]	2021	2021	12313	MT for processor verification	C++	No license	RISC-V	✓	✗	✗	95	✗	Command line	Text	✓	✓
MT-EA4Cloud [9]	2020	2021	5842	Optimisation of cloud systems by combining MT and EAs	Java	No license	Cloud computing	✓	✓	✓	8	✗	Command line	Text	✓	✓
FwCloudMeT [27]	2022	2022	5740	Selection of the most suitable simulator covering the features of interest for the user	Java	No license	Cloud computing	✓	✓	✗	6	✗	GUI	Panel	✓	✓

MRs that can be defined with MTKeras, MT4WS and CheckerDSL target their respective domains (machine learning models, Web services, and programs with numeric inputs). Instead, our DSL supports defining input features with complex structure (connected objects) in OCL. Moreover, Gotten features extension points to facilitate the definition of environments for arbitrary domains, while the other tools have a fixed application domain.

The last block considers the results of the MT process. MT generates a large amount of information that the tester must analyse, like the follow-ups that do not satisfy the MRs, or the value of the input/output features. The first issue is relevant to investigate the cause of misbehaviours. Moreover, the statistics of the process provide interesting information to the tester. Therefore, MT solutions should provide effective mechanisms to display and manage this information.

Overall, five applications provide a GUI to manage the MT process, and nine are controlled via the command line. Four of these applications show the testing results graphically, while the rest use plain text. Gotten provides an intuitive and well-organised GUI to manage the testing results (see Fig. 8). This information can be navigated and expanded to show the values involved in the execution of each follow-up, the MRs that are not satisfied by the follow-ups, and the results of each individual test case. On the contrary, only four applications show information about the follow-ups that pass or fail, and five applications report just the percentage of test cases that satisfy each MR.

In conclusion, we observe that Gotten generates MT environments with functionality up to par with existing MT solutions. It is one of the few approaches that can be extended with new MRs using a DSL, supports the configuration of the follow-up generation process, and provides detailed, interactive reports. Most importantly, while all the other tools are specific to a particular domain, Gotten can be adapted to different domains.

8.3. Approaches combining MT and MDE

Since Gotten is an MDE solution for MT, we next review approaches combining both fields, although it is an area barely investigated. MT has been used to test model transformations [48–50] and code generators [51]. Boussaa et al. [51] detect inconsistencies in code generators using non-functional MRs for resource usage and performance. Troya et al. [52] automatically infer MRs in ATL transformations based on the execution traces. They define 24 domain-independent MRs, which are used as a basis for instantiating new MRs for transformation programs.

Hence, we observe that MT has been applied to test MDE artefacts. However, to our knowledge, no approach has used MDE to create MT environments. A framework like ours can facilitate building MT environments for all sorts of MDE artefacts.

9. Conclusions and future work

In this paper, we have presented a model-driven solution called Gotten to automate the construction of MT environments for any domain. The approach offers two DSLs to describe MRs, strategies for generating follow-ups, and an extension point to connect the MT environment with the SuT. We have illustrated its effectiveness by creating an MT environment for testing data centre simulators and comparing the effort w.r.t. other ad-hoc tools. We have evaluated the suitability of the tool by replicating previous experiments, and its generality by defining an MT environment for video streaming APIs.

In the future, we plan to improve the expressivity of *fowDSL* by adding application conditions and other types of mutation operations. We will also apply our framework to other domains, both within MDE (e.g., to test model transformations or code generators) and in other areas. Finally, we would like to conduct a user evaluation to better understand the strong and weak points of the proposal from the user perspective.

CRediT authorship contribution statement

Pablo Gómez-Abajo: Software, Validation, Writing – original draft, Review & editing. **Pablo C. Cañizares:** Conceptualization, Software, Validation, Writing – original draft, Review & editing. **Alberto Núñez:** Conceptualization, Validation, Writing – original draft, Review & editing, Funding acquisition. **Esther Guerra:** Conceptualization, Methodology, Writing – original draft, Review & editing, Funding acquisition. **Juan de Lara:** Conceptualization, Methodology, Writing – original draft, Review & editing, Supervision, Funding acquisition.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.infsof.2023.107164.

Data availability

Data will be made available on request.

Acknowledgements

Work supported by the Madrid government, Spain and the Complutense University (grant PR65/19-22452), the Spanish Ministry of Science (PID2021-122270OB-I00), and the Madrid region, Spain (P2018/TCS-4314, S2018/TCS-4339).

References

[1] T.Y. Chen, S.C. Cheung, S.M. Yiu, Metamorphic testing: A new approach for generating next test cases, Technical report, HKUST-CS98-01, 1998.
[2] S. Segura, D. Towey, Z.Q. Zhou, T.Y. Chen, Metamorphic testing: Testing the untestable, IEEE Softw. 37 (3) (2020) 46–53.
[3] E. Weyuker, On testing non-testable programs, Comput. J. 25 (4) (1982) 465–470.
[4] T.Y. Chen, F. Kuo, H. Liu, P. Poon, D. Towey, T.H. Tse, Z.Q. Zhou, Metamorphic testing: A review of challenges and opportunities, ACM Comput. Surv. 51 (1) (2018) 4:1–4:27.
[5] S. Segura, G. Fraser, A.B. Sánchez, A. Ruiz Cortés, A survey on metamorphic testing, IEEE Trans. Softw. Eng. 42 (9) (2016) 805–824.
[6] Y. Deng, X. Zheng, T. Zhang, G. Lou, M. Kim, RMT: Rule-based metamorphic testing for autonomous driving models, 2020, arXiv preprint arXiv:2012.10672.
[7] M. Mansur, M. Christakis, V. Wüstholz, Metamorphic testing of Datalog engines, in: ESEC-FSE, 2021, pp. 639–650.
[8] F. Riese, V. Herdt, D. Große, R. Drechsler, Metamorphic testing for processor verification: A RISC-V case study at the instruction level, in: VLSI-SoC, IEEE, 2021, pp. 1–6.
[9] P.C. Cañizares, A. Núñez, J. de Lara, L. Llana, MT-EA4Cloud: A methodology for testing and optimising energy-aware cloud systems, J. Syst. Softw. 163 (2020) 110522.
[10] Y. Liu, Z. Zhou, T. Chen, Y. Liu, D. Towey, MTKeras: An automated metamorphic testing platform, IJSEKE 31 (09) (2021) 1235–1249.
[11] M. Zhang, Y. Zhang, L. Zhang, C. Liu, S. Khurshid, Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems, in: ASE, ACM, 2018, pp. 132–142.
[12] B. Combemale, R. France, J.-M. Jézéquel, B. Rumpe, J. Steel, D. Vojtisek, Engineering modeling languages. Turning domain knowledge into tools, Chapman and Hall/CRC, 2017.
[13] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, Second Edition, in: Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, 2017.
[14] M. Voelter, DSL Engineering - Designing, Implementing and Using Domain-Specific Languages, dslbook.org, ISBN: 978-1-4812-1858-0, 2013, URL http://www.dslbook.org.
[15] S. Segura, J.A. Parejo, J. Troya, A. Ruiz Cortés, Metamorphic testing of RESTful web APIs, IEEE Trans. Software Eng. 44 (11) (2018) 1083–1099.
[16] P.C. Cañizares, P. Gómez-Abajo, A. Núñez, E. Guerra, J. de Lara, New ideas: Automated engineering of metamorphic testing environments for domain-specific languages, in: SLE, ACM, 2021, pp. 49–54.
[17] R. Bill, M. Fleck, J. Troya, T. Mayerhofer, M. Wimmer, A local and global tour on MOMoT, Softw. Syst. Model. 18 (2) (2019) 1017–1046.

[18] A. Núñez, P.C. Cañizares, M. Núñez, R.M. Hierons, TEA-Cloud: A formal framework for testing cloud computing systems, IEEE Trans. Reliability 70 (1) (2021) 261–284.
[19] X. Lin, M. Simon, N. Niu, Exploratory metamorphic testing for scientific software, Comput. Sci. Eng. 22 (2) (2020) 78–87.
[20] X. Xie, J.W.K. Ho, C. Murphy, G.E. Kaiser, B. Xu, T.Y. Chen, Testing and validating machine learning classifiers by metamorphic testing, J. Syst. Softw. 84 (4) (2011) 544–558.
[21] V. Le, M. Afshari, Z. Su, Compiler validation via equivalence modulo inputs, in: PLDI, ACM, 2014, pp. 216–226.
[22] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exper. (ISSN: 0038-0644) 41 (1) (2011) 23–50.
[23] H. Ouarnoughi, J. Boukhobza, F. Singhoff, S. Rubini, Integrating I/Os in Cloudsim for performance and energy estimation, Oper. Syst. Rev. 50 (1) (2017) 27–36.
[24] G. Kecskemeti, DISSECT-CF: A simulator to foster energy-aware scheduling in infrastructure clouds, Simul. Model. Pract. Theory 58 (2015) 188–218.
[25] Object Management Group, UML 2.4 OCL Specification, 2014, http://www.omg.org/spec/OCL/.
[26] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework, 2^{Nd} Edition, Addison-Wesley Professional, 2008.
[27] A. Núñez, P.C. Cañizares, J. de Lara, CloudExpert: An intelligent system for selecting cloud system simulators, Expert Syst. Appl. 187 (2022) 115955.
[28] M. Harman, S.A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (1) (2012).
[29] N. Przigoda, P. Niemann, J.G. Filho, R. Wille, R. Drechsler, Frame conditions in the automatic validation and verification of UML/OCL models: A symbolic formulation of modifies only statements, Comput. Lang. Syst. Struct. 54 (2018) 512–527.
[30] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Fundamentals of Algebraic Graph Transformation, Springer, 2006.
[31] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer, Henshin: Advanced concepts and tools for in-place EMF model transformations, in: MODELS, in: LNCS, vol. 6394, Springer, 2010, pp. 121–135.
[32] R. Clarisó, J. Cabot, E. Guerra, J. de Lara, Backwards reasoning for model transformations: Method and applications, J. Syst. Softw. 116 (2016) 113–132.
[33] Xtext, 2022, http://www.eclipse.org/Xtext/, (Accessed December 2022).
[34] K. Alpernas, Y.M.Y. Feldman, H. Peleg, The wonderful wizard of LoC: paying attention to the man behind the curtain of lines-of-code metrics, in: Onward!, ACM, 2020, pp. 146–156.
[35] C. ai Sun, G. Wang, B. Mu, H. Liu, Z. Wang, T.Y. Chen, Metamorphic testing for web services: Framework and a case study, in: ICWS, IEEE, 2011, pp. 283–290.
[36] C. ai Sun, G. Wang, Q. Wen, D. Towey, T.Y. Chen, MT4WS: An automated metamorphic testing system for web services, Int. Journal High Perf. Comp. and Networking 9 (1/2) (2016) 104–115.
[37] FogMET, 2021, https://github.com/oceanao/FogMET.
[38] A. Gotlieb, B. Botella, Automated metamorphic testing, in: COMPAC, IEEE, 2003, pp. 34–40.
[39] R.I. Hadiwijaya, M.M. Liem, Metamorphic testing and DSL for test cases & checker generators, Olympiads Inform. 9 (2015) 75–88.
[40] A. Sharma, H. Wehrheim, Testing machine learning algorithms for balanced data usage, in: ICST, IEEE, 2019, pp. 125–135.
[41] Z. Zhang, P. Wang, H. Guo, Z. Wang, Y. Zhou, Z. Huang, DeepBackground: Metamorphic testing for deep-learning-driven image recognition systems accompanied by background-relevance, Inf. Softw. Technol. 140 (2021) 106701.
[42] D. Cao, H. Guo, C. Tao, MT4ImgRec: A metamorphic testing tool for image recognition software, in: SEKE, KSI research institute, 2021, p. paper 205.
[43] H. Spieker, A. Gotlieb, Adaptive metamorphic testing with contextual bandits, J. Syst. Softw. 165 (2020) 110574.
[44] Y. Yang, T.F.M. Chui, Reliability assessment of machine learning models in hydrological predictions through metamorphic testing, Water Resour. Res. 57 (9) (2021) e2020WR029471.
[45] M. Asyrofi, Z. Yang, I. Yusuf, H. Kang, F. Thung, D. Lo, Biasfinder: Metamorphic test generation to uncover bias for sentiment analysis systems, IEEE Trans. Software Eng. In press (2021).
[46] P. Ma, S. Wang, J. Liu, Metamorphic testing and certified mitigation of fairness violations in NLP models., in: IJCAI, 2020, pp. 458–465.
[47] J. Almendros-Jiménez, A. Becerra-Terón, M. Merayo, M. Núñez, Metamorphic testing of OpenStreetMap, Inf. Softw. Technol. 138 (2021) 106631.
[48] K. Du, M. Jiang, Z. Ding, H. Huang, T. Shu, Metamorphic testing in fault localization of model transformations, in: SOFL+MSVL, Springer, 2019, pp. 299–314.
[49] X. He, X. Chen, S. Cai, Y. Zhang, G. Huang, Testing bidirectional model transformation using metamorphic testing, Inf. Softw. Technol. 104 (2018) 109–129.
[50] M. Jiang, T.Y. Chen, F. Kuo, Z. Zhou, Z. Ding, Testing model transformation programs using metamorphic testing, in: SEKE, KSI, 2014, pp. 94–99.
[51] M. Boussaa, O. Barais, G. Sunyé, B. Baudry, Leveraging metamorphic testing to automatically detect inconsistencies in code generator families, Softw. Test. Verification Reliab. 30 (1) (2020).
[52] J. Troya, S. Segura, A. Ruiz Cortés, Automated inference of likely metamorphic relations for model transformations, J. Syst. Softw. 136 (2018) 188–208.