Science of Computer Programming 230 (2023) 103000

Contents lists available at ScienceDirect

Science of Computer Programming

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

Original software publication

Wodel-Edu: A tool for the generation and evaluation of diagram-based exercises

Pablo Gómez-Abajo*, Esther Guerra, Juan de Lara

Computer Science Department, Universidad Autónoma de Madrid, Madrid, Spain

ARTICLE INFO

Article history:
Received 8 December 2022
Received in revised form 7 July 2023
Accepted 10 July 2023
Available online 13 July 2023

Keywords:
Model-driven engineering
Education
Model mutation
Automated generation of exercises
Moodle

ABSTRACT

Creating and grading exercises are recurring tasks within higher education. When these exercises are based on diagrams – like logic circuits, automata or class diagrams – we can represent them as models, and use model-driven engineering techniques for the large-scale generation of quizzes, which can be automatically graded.

This way, we propose a domain-independent tool for the generation and automated evaluation of diagram-based exercises called Wodel-Edu. Wodel-Edu is built atop Wodel, an extensible tool for model mutation, and offers seven kinds of diagram exercises. It supports code generation from the exercises for the Moodle platform, the web, Android and iOS applications. Evaluations from the professor and student perspectives show good results.

© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Code metadata

Code metadata description	
Current code version	1.2
Permanent link to code/repository used of this code version	https://github.com/ScienceofComputerProgramming/SCICO-D-22-00339
Permanent link to Reproducible Capsule	https://gomezabajo.github.io/Wodel/wodel-edu-video.html
Legal Code License	EPL-1.0 License
Code versioning system used	git
Software code languages, tools, and services used	Eclipse Modelling Tools 2023-03 Release (4.27), EMF 2.33.0, Java 11, Xtext 2.30.0, Xtend 2.30.0, Sirius 7.1.0, EMF Compare 3.5.3, OCL Examples and Editors SDK 6.18.0, USE ModelValidator 4.2.0, Graphviz 8.0.3, Circuit Macros 10.0.2, emfjson 0.13.0
Compilation requirements, operating environments & dependencies	Microsoft Windows 7 64-bit or later, Linux
If available Link to developer documentation/manual	https://gomezabajo.github.io/Wodel/wodel-edu.html
Support email for questions	pablo.gomeza@uam.es

1. Motivation and significance

One of the effects of the global pandemic has been the vast increasing need in higher education for a transition from traditional face-to-face teaching into a long-distance or hybrid model. This long-distance educational model requires a mas-

* Corresponding author.
E-mail addresses: Pablo.GomezA@uam.es (P. Gómez-Abajo), Esther.Guerra@uam.es (E. Guerra), Juan.deLara@uam.es (J. de Lara).

https://doi.org/10.1016/j.scico.2023.103000
0167-6423/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

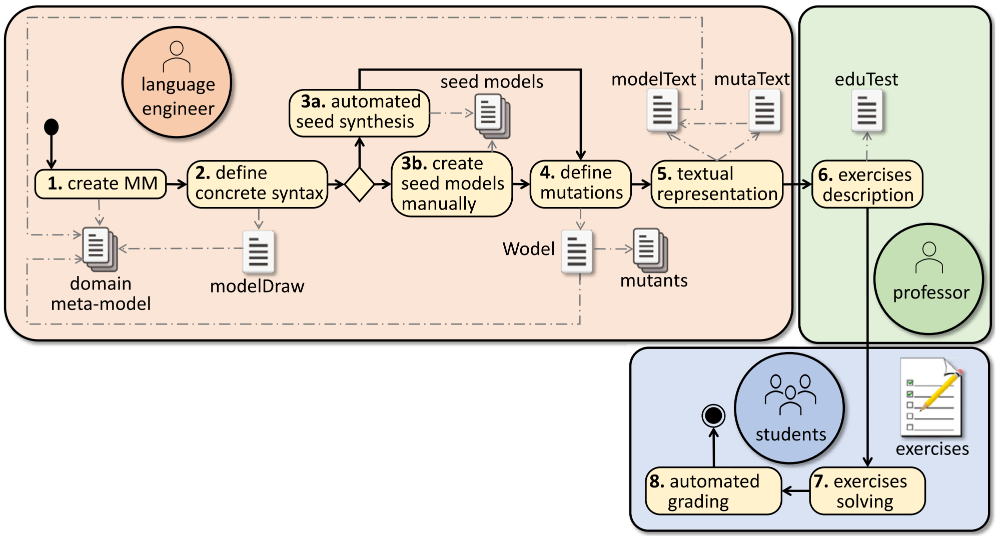


Fig. 1. WODEL-EDU process.

sive creation of digital resources, such as course notes, videos, and exercises [12]. The task of a fair online (self-)assessment of students becomes very frequently an unreachable effort for the professors, who need to create large amounts of exercise variations and their evaluation in very short periods of time.

When these exercises are based on diagrams – such as those in the domains of software design, logic circuits, automata theory or chemistry – we can represent them as models. Hence, we can use the automation techniques provided by model-driven engineering (MDE) to generate exercises and provide their automated evaluation [4].

Following this idea, we propose a model-based solution called WODEL-EDU, which represents the exercises as models. WODEL-EDU supports the generation of exercises for any diagram-based domain, based on the automated generation of incorrect answers (models) from correct ones, which enables fully accurate grades. In addition, WODEL-EDU allows the user to configure the graphical and textual representation of the models by means of a family of DSLs, and the selection of the kind of exercise that best fits the corresponding assignment among a set of seven.

2. Software description

Fig. 1 illustrates the process followed by WODEL-EDU to generate exercises. It involves the roles of *language engineer* and *professor*, which can be played by the same person. The former defines the modeling language over which the exercises are created. The latter describes the text of the exercises and selects the types of exercises to be generated. Then, *students* can solve the generated exercises, which get an automated grading.

In the first step, the *language engineer* must create the domain meta-model if it does not exist, and define its graphical concrete syntax. As WODEL-EDU generates diagrams for different platforms (MOODLE, the web, ANDROID or IOS), WODEL-EDU uses a dedicated language to specify such concrete syntax, called MODELDRAW.

Next, she facilitates an initial set of models, upon which the exercises are based. This task can be performed either manually (step *3b*) or using the automated seed model synthesizer provided by the WODEL-EDU tool (step *3a*).

Then, the *language engineer* defines the mutation operators of interest to generate the mutants. These operators specify minimum model modifications, producing variations in the initial set of models. These are used to generate the exercises, which are test-based (i.e., answers are based on a closed set of options). For example, in the simplest exercise type, the student must distinguish correct (i.e., from the initial model set) from incorrect models (i.e., a model mutant). To define and execute the mutation operations, WODEL-EDU uses a DSL called WODEL.

In the fifth step, the *language engineer* provides the textual representation of the models and the applied mutation operators, using the MODELTEXT and MUTATEXT DSLs. This is required by some types of exercises, which need the textual description either of parts of the model or the modifications performed.

At this point, the process is ready for the *professor* to define and customize the set of exercises to be generated. This is done using the EDUTEST DSL. Finally, the set of exercises is generated and ready to evaluate the *students*.

Fig. 2 shows the seven kinds of exercises currently supported by WODEL-EDU. Some exercise kinds require a textual statement, and the *professor* can benefit from the extension point provided by WODEL-EDU to obtain an alternative textual representation of the models and speed up the statements creation process. The exercises supported are:

- *Alternative Response.* This kind of exercise shows a diagram that either corresponds to the textual statement, i.e., the seed model – where the answer is *true*, or one of its mutants – where the answer is *false* (see Fig. 2a).

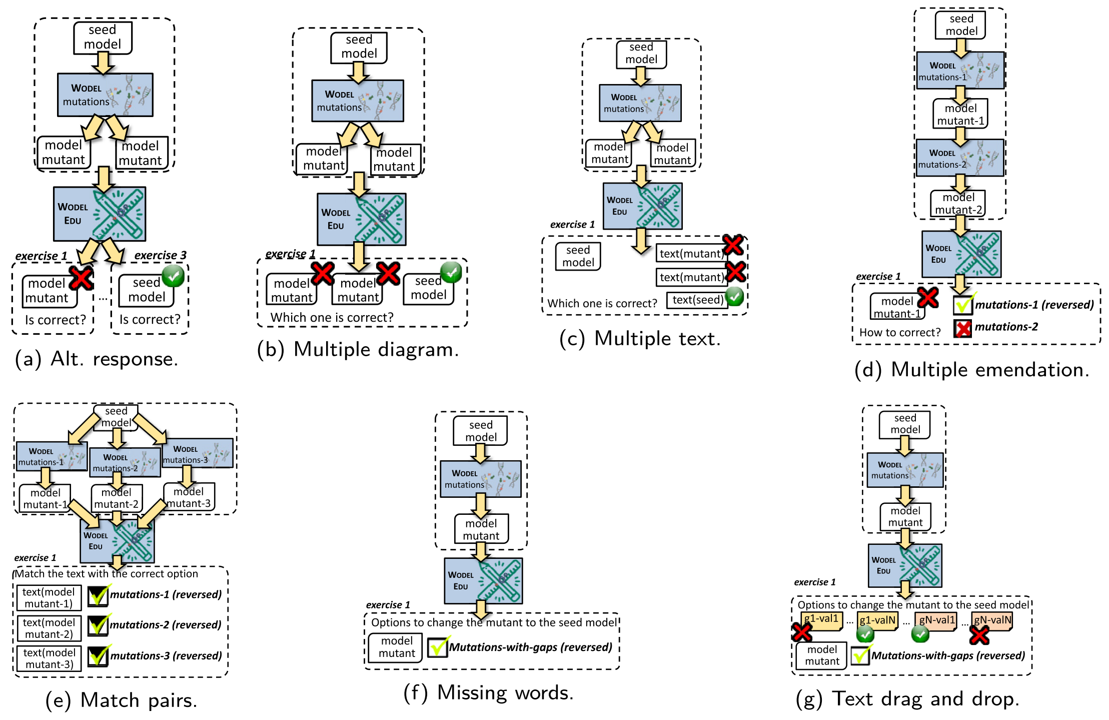


Fig. 2. Supported kinds of exercises.

- *Multiple Diagram Choice.* In this case, the exercise presents a textual statement and a set of diagrams that includes the one corresponding to the seed model – the rest corresponds to other generated mutants. The student must tell which of these diagrams corresponds to the statement, i.e., the seed model (see Fig. 2b).
- *Multiple Text Choice.* This kind of exercise shows a diagram and a set of textual representations of the models. One of these provided textual representations corresponds to the statement, i.e., the seed model, and the remaining corresponds to other generated mutants. In this case, the solution is the textual representation of the seed model (see Fig. 2c).
- *Multiple Emendation Choice.* This exercise presents a diagram that corresponds to a mutant, a textual statement, and a set of textual options to emend it. The textual options correspond to a set of mutation operators that can be applied to the shown diagram. The options that fix the diagram to satisfy the provided textual statement, i.e., the textual representation of the seed model, are the correct answers. The remaining options, when applied to the shown mutant, generate a different model (see Fig. 2d).
- *Match Pairs.* In this kind of exercise, the student must match a set of textual representations of mutation operators with a set of textual representations of the generated mutants after they are applied to the shown diagram (see Fig. 2e).
- *Missing Words.* This exercise shows a textual statement that corresponds to the seed, a diagram that corresponds to one of its mutants, and a set of textual representations of the mutation operators used to generate the presented diagram. Each of these textual representations includes a gap for each variable field that contains a drop-down list with all its possible values. In this case, the student must fill the gaps by selecting from such drop-down lists the value taken by the corresponding mutation operator to generate the shown mutant (see Fig. 2f).
- *Text Drag and Drop.* The process followed to create this kind of exercise is similar to the one used in the *Missing Words* kind. In this case, the missing text values are shown in labels grouped by category and the student can drag and drop these labels to each gap. The solution to this exercise is the combination of labels in each gap that matches the values taken by the applied mutation operators (see Fig. 2g).

3. Software architecture

Wodel-Edu is an Eclipse plugin, available at: http://gomezabajo.github.io/Wodel/wodel-edu.html.

Fig. 3 shows the architecture of Wodel-Edu. Its engine provides editors for a family of five DSLs that allow the configuration, specification and generation of the exercise applications. As later explained in Section 4, the first of these DSLs is Wodel, a domain-independent language for model mutation that we presented in [8].

First, the *language engineer* specifies the domain meta-model (label 1) and includes a set of seed models over which the selected mutation operators encoded in Wodel are applied (label 2). Alternatively, Wodel-Edu supports an automated synthesis of the seed models (label 3), over which the defined mutation operators are ensured to be applicable, and allows

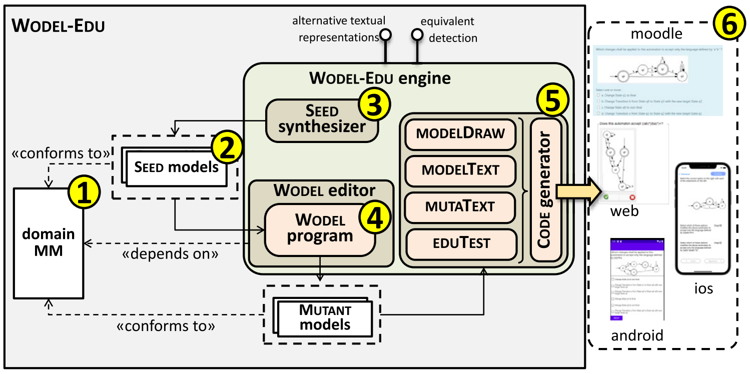
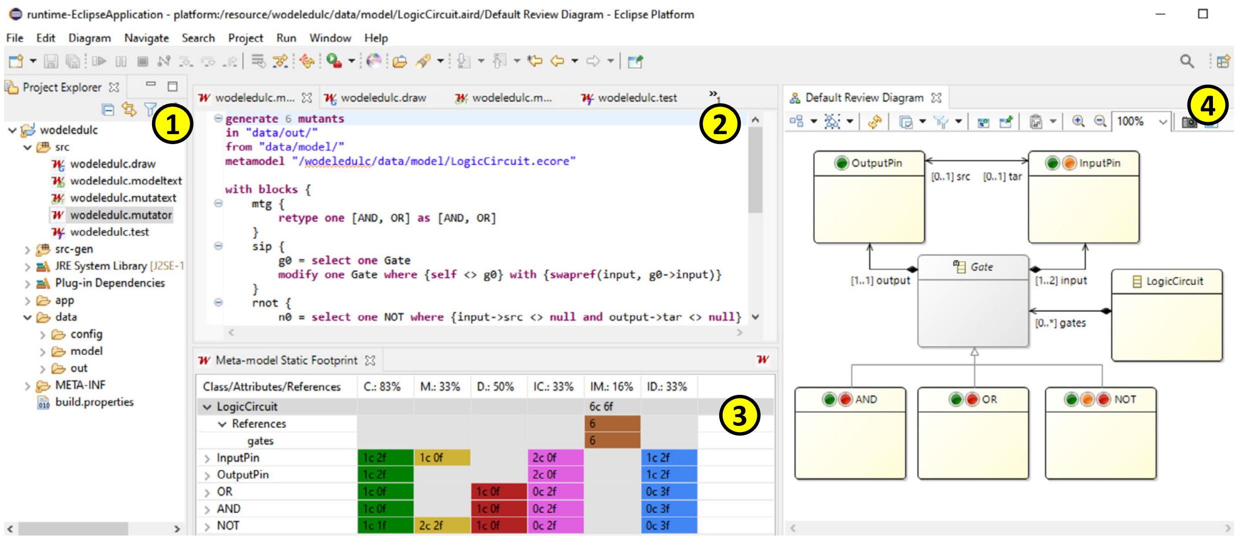


Fig. 3. Architecture of WODEL-EDU.



```
generate 6 mutants
in "data/out/"
from "data/model/"
metamodel "/wodeledulc/data/model/LogicCircuit.ecore"

with blocks {
    mtg {
        retype one [AND, OR] as [AND, OR]
    }
    sip {
        g0 = select one Gate
        modify one Gate where {self <> g0} with {swapref(input, g0->input)}
    }
    rnot {
        n0 = select one NOT where {input->src <> null and output->tar <> null}
```

Meta-model Static Footprint

Class/Attributes/References	C.: 83%	M.: 33%	D.: 50%	IC.: 33%	IM.: 16%	ID.: 33%
LogicCircuit					6c 6f	
References					6	
gates					6	
InputPin	1c 2f	1c 0f		2c 0f		1c 2f
OutputPin	1c 2f			2c 0f		1c 2f
OR	1c 0f		1c 0f	0c 2f		0c 3f
AND	1c 0f		1c 0f	0c 2f		0c 3f
NOT	1c 1f	2c 2f	1c 0f	0c 2f		0c 3f

Fig. 4. The WODEL-EDU environment.

generating exercises of different difficulty levels. For this purpose, WODEL-EDU relies on the USE model finder [11]. Based on this specification, this user generates a set of mutants by means of the WODEL tool (label 4). In the next step (label 5), the *language engineer* specifies how the models are graphically represented with the MODELDRAW DSL. The MODELTEXT DSL allows specifying the textual representation of the elements of the models, and the MUTATEXT DSL permits the definition of the textual representation of the applied mutation operators.

Then, the *professor* can configure the features of the exercise application to be generated, such as the kinds of exercises, or whether the application enables reattempts or not, by means of the EDUTEST DSL. The provided code generator currently supports the generation of exercises for MOODLE, the web, and ANDROID and IOS applications (label 6).

In addition, WODEL-EDU provides two extension points. The first one allows the definition of an alternative (complex) textual representation for the models. As Section 4 shows, the *language engineer* can use this extension point to represent a digital circuit with its equivalent boolean expression, or an automaton with its equivalent regular expression. The second extension point facilitates the detection of equivalent mutants. This is useful to detect if two artifacts do not differ semantically after applying a mutation operator. Since the exercises assume that mutants are semantically different from the original model, these are discarded. Note that the tool avoids generating mutants that do not conform to the meta-model and its OCL invariants.

Fig. 4 shows the WODEL-EDU development environment. Label 1 is the project explorer, with the set of programs to configure and generate the exercises. Label 2 shows the WODEL editor – WODEL-EDU provides editors for the MODELDRAW, MODELTEXT, MUTATEXT, and EDUTEST DSLs, too (not shown). Label 3 shows statistics on the number of times each meta-model element is affected by the application of the mutation operators. Label 4 represents this meta-model coverage by the mutation operators graphically. Both label 3 and label 4 views allow the *language engineer* to understand the behavior of the mutation operators over the models.

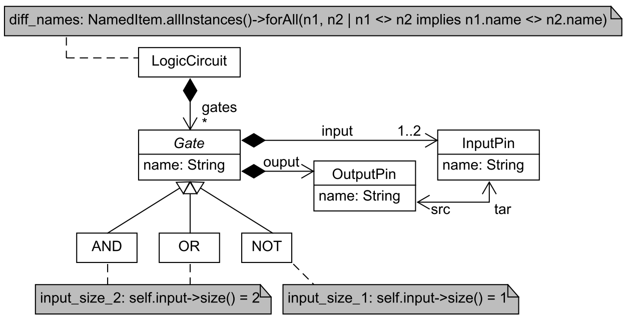
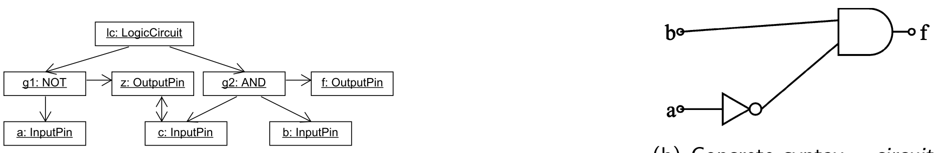

diff_names: NamedItem.allInstances()->forAll(n1, n2 | n1 <> n2 implies n1.name <> n2.name)
LogicCircuit
gates
*
Gate
name: String
input
1..2
InputPin
name: String
ouput
OutputPin
name: String
src
tar
AND
OR
NOT
input_size_2: self.input->size() = 2
input_size_1: self.input->size() = 1

Fig. 5. Logic circuit meta-model.



(a) Abstract syntax.

(b) Concrete syntax – *circuit macros* notation.

Fig. 6. Logic circuit model in its abstract syntax on the left and concrete syntax on the right.

```
 1 generate exhaustive mutants in "data/out/" from "data/model/"
 2 metamodel "http://lc/1.0"
 3
 4 with blocks {
 5   mtg1 "retypes an and gate as an or gate" {
 6     retype one AND as OR
 7   } [3]
 8   mtg2 from mtg1 repeat=no
 9   "retypes an or gate as an and gate from mutants generated by mtg1" {
10     retype one OR as AND
11   } [3]
12 }
```

Listing 1: Simple Wodel program.

4. Illustrative examples

As an illustration, we generate exercises for logic circuits (LCs). Fig. 5 shows the meta-model for this notation. An LC consists of a set of Gates, which have a name, and are composed of a set of one or two InputPins and an OutputPin. An InputPin has a name and a reference src to an OutputPin, and an OutputPin has a name and a reference tar to an InputPin. A Gate can be of type AND, OR or NOT.

The meta-model has OCL constraints to require that all the elements in an LC have different names (diff_names), the AND and OR Gates have two InputPins (input_size_2) and NOT Gates have one InputPin (input_size_1).

Fig. 6 presents a model conforming to the meta-model in Fig. 5, in its abstract syntax on the left (Fig. 6a), and its concrete syntax using the Circuit Macros[1] notation on the right (Fig. 6b). This LC example has two initial InputPins (named 'a' and 'b'), a NOT gate with the InputPin 'a' as its input, and an AND gate with inputs the InputPin 'b' and the output of the NOT gate, and with output the OutputPin 'f'.

Next, we exemplify the use of the five DSLs to specify and generate LC exercises.

4.1. *Defining mutations with* Wodel

The Wodel DSL supports the definition of mutation operators for any model defined by a meta-model [8]. It provides a set of nine mutation primitives to select, create, clone, modify, retype (i.e., modify the type of an object to one of its sibling types) and remove objects, and to create, modify and remove references.

Listing 1 shows a Wodel program that defines two mutation operators for LCs. Line 1 declares the strategy for mutant generation, which in this case is exhaustive, meaning the generation of all possible mutants for the given mutation operators. Alternatively, Wodel provides a stochastic mode, where the *language engineer* inputs the maximum number of mutants to

[1] https://ece.uwaterloo.ca/~aplevich/Circuit_macros/.

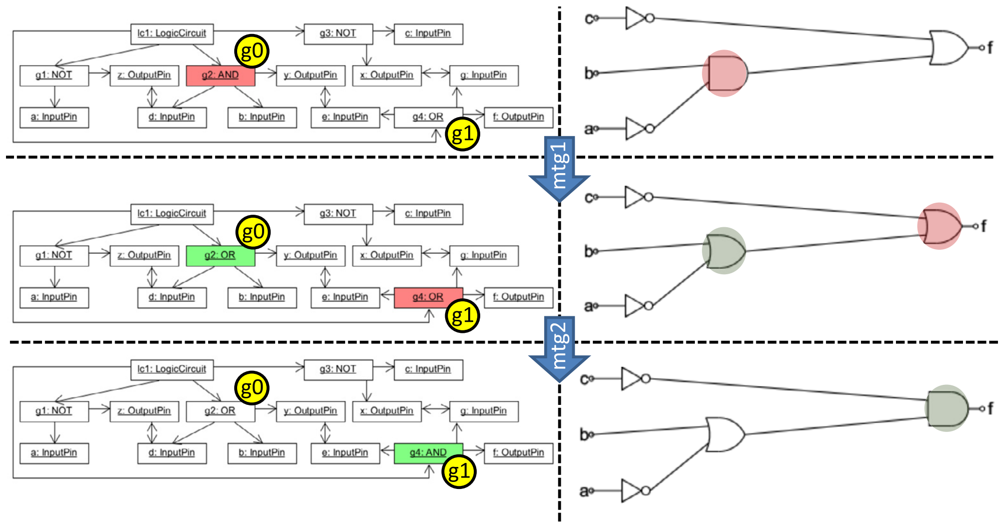


Fig. 7. Example application of the mtg1 and mtg2 operators. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

```
1  metamodel "http://lc/1.0"
2
3  LogicCircuit: diagram {
4     Gate: node shape=logic
5     InputPin(src == null): node shape=circle
6     OutputPin(tar == null): node shape=circle
7     InputPin(src == null) →Gate: edge
8     Gate →OutputPin(tar == null): edge
9     Gate(output→tar) →Gate: edge
10 }
```

Listing 2: modelDraw excerpt.

generate. Line 1 also specifies the output folder to store the mutants and the input folder containing the seed models. Line 2 declares the URI or location of the language meta-model. The rest of the program defines two mutation operators to modify the type of, i.e., to *retype*, AND and OR gates. Lines 5–7 define the mutation operator mtg1 to retype an AND gate as an OR gate. Line 7 shows the Wodel functionality to limit the maximum number of mutants to be generated for each mutation operator, in this case, 3. Lines 8–11 declare a similar mutation operator to mtg1, in this case to retype an OR gate as an AND gate, taking the mutants generated by mtg1 as the seed models and requiring the generated mutants to be different from the previous ones with the directive repeat=no. Note this directive can be beneficial when we have mutation operators of higher orders.

Fig. 7 shows an example application of the defined mtg1 and mtg2 operators to an LC with its abstract syntax on the left and its concrete syntax on the right. The model elements affected by the operator execution are labeled as g0 and g1. The seed model of the LC is presented in the first row, where the AND gate g0 shaded in red is selected to be retyped as an OR gate by the mutation operator mtg1. The resulting mutant is shown in the second row, where the new OR gate g0 is shaded in green. Now, the mtg2 operator takes action, and the OR gate labeled as g1 and shaded in red in the second row is selected to be retyped as an AND gate. The third row presents the resulting mutant with the new AND gate g1 created by the mtg2 operator shaded in green.

4.2. *Defining the graphical syntax with* ModelDraw

The modelDraw DSL allows configuring the graphical representation of the models. It permits attaching different shapes to the meta-model classes, and using conditional styles depending on the feature values.

Listing 2 shows the modelDraw specification for LCs. The first line declares the language meta-model. Next, the listing declares that gates are represented using their corresponding logic node shape for the three kinds of gates included – AND, OR and NOT – (line 4), and the InputPins that do not have an src OutputPin, i.e., initial InputPins, and the OutputPins that do not have a tar InputPin, i.e., final OutputPins, as circles (lines 5–6). Finally, the relations between the initial InputPins and Gates (line 7), Gates and final OutputPins (line 8), and non-final Gates with their immediate connected Gates (line 9) are represented as edges.

```
>Gate: %type gate
>Gate(input→src == null): input
>Gate(output→tar == null): output
>InputPin: input pin %name
>OutputPin: output pin %name
>InputPin.src: source
>OutputPin.tar: target
```

Listing 3: MODELTEXT excerpt.

```
>ObjectRetyped: Change %object to %toObject /
  Change %toObject to %object
```

Listing 4: MUTATEXT program.

The WODEL-EDU tool provides two kinds of graphical representations for the diagrams, enabling the *language engineer* to select from the preferences page which of them best fits each domain. The first one is more generic and uses the Graphviz[2] notation. The second uses the Circuit Macros library, and is specific for electric, electronic and line circuits, as illustrated in Fig. 6b.

4.3. *Representing model elements textually with* MODELTEXT

The text options for the exercises *multiple emendation choice*, *match pairs*, *missing words*, and *text drag and drop* need to represent the model elements textually.

The MODELTEXT DSL takes by default either the attribute name of the element, if it exists, or the class name as its textual description. Additionally, this DSL allows overriding this text representation for the meta-model classes and relations. This is achieved by providing a text template where the expressions preceded by % are evaluated over the element, and print their value in the resulting description text.

Listing 3 shows the MODELTEXT specification for LCs. Line 1 specifies that if an object of type Gate is used in some text, it will be written as its type – AND, OR, or NOT – followed by “gate”. For example, “AND gate” if the gate is of type AND. Lines 2–3 define prefixes for the gates descriptions depending on the value of its references input->src and output->tar. Hence, a gate of type AND that has an OutputPin with no tar as its output reference, is textually represented as “output AND gate”. If an element is both input and output, both prefixes are conjoined. Line 4 defines that objects of type InputPin should be represented with the text “input pin” followed by their name, e.g., an InputPin named “a” is textually represented as “input pin a”. Line 5 defines the analogous description for OutputPin elements. Finally, lines 6–7 declare that the references src of InputPins and tar of OutputPins are to be referred to as “source” and “target”, respectively.

4.4. *Describing the mutations with* MUTATEXT

Mutation operators also need to be represented as text. This is required to create the list of options in the *multiple emendation choice*, *match pairs*, *missing words*, and *text drag and drop* exercises.

WODEL-EDU has default textual templates to represent each mutation primitive. These templates contain placeholders for the textual representation of the elements they are applied to. For example, the default template for the retype object primitive is ‘Change <objectClassName> <objectName> to <toObjectClassName> <toObjectName>’. MUTATEXT allows overriding these default templates.

Listing 4 shows the MUTATEXT specification for LCs. It defines the text to represent the retype object operator when it is displayed in a correct answer of an exercise (line 1) and when it is a wrong answer (line 2). In addition, it enables using predefined variables that contain information about the applied mutation, such as %object, which identifies the mutated object. These variables return the textual representation of the object or reference specified with MODELTEXT, or a default textual representation if this specification is not available. For example, the text defined in line 2 of Listing 4 in combination with the MODELTEXT definition of Listing 3 generates text options such as: “Change input AND gate to input OR gate”.

4.5. *Describing the exercises with* EDUTEST

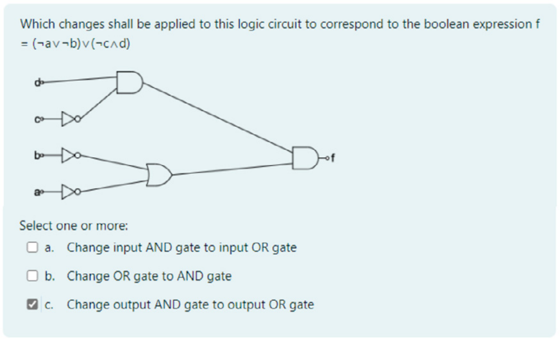
The EDUTEST DSL allows describing and configuring the set of exercises to generate. As explained in Section 2, WODEL-EDU currently supports seven kinds of exercises.

This DSL enables the *professor* to configure the aspects of the exercises: kind (*alternative response*, *multiple diagram choice*, *multiple text choice*, *multiple choice emendation*, *match pairs*, *missing words*, and *text drag and drop*), order, statement, and retry options. From this specification, WODEL-EDU generates the exercises for the target environment (currently, MOODLE, the web, and ANDROID and IOS applications).

[2] http://www.graphviz.org/.

```
 1  navigation=free
 2  MultiChoiceEmendation mtg1 {
 3    retry=no, weighted=no, penalty=0.0,
 4    order=options−descending, mode=checkbox
 5    description for 'lc1.model' =
 6      'Which changes shall be applied to this logic circuit to correspond to the boolean expression ' %text('bool−exp')
 7    description for 'lc2.model' =
 8      'Which changes shall be applied to this logic circuit to correspond to the boolean expression ' %text('bool−exp')
 9  }
10  MatchPairs mtg1, mtg2, mtg3 {
11    retry=no, text='bool−exp'
12    description for 'lc3.model' =
13      'Match the correct option on the right that modifies the below logic circuit to correspond with each of the boolean expressions on the left'
14    description for 'lc4.model' =
15      'Match the correct option on the right that modifies the below logic circuit to correspond with each of the boolean expressions on the left'
16  }
```

Listing 5: Simple eduTest program.

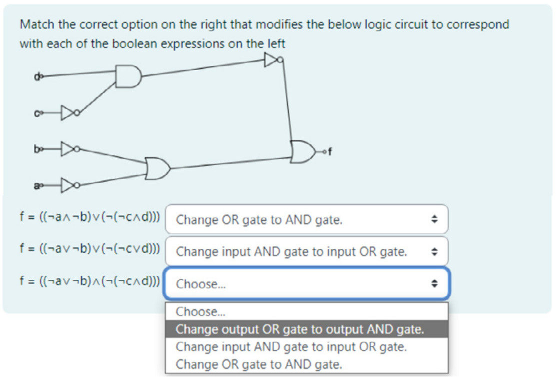


Which changes shall be applied to this logic circuit to correspond to the boolean expression f = (¬a∨¬b)∨(¬c∧d)

Select one or more:

- [ ] a. Change input AND gate to input OR gate
- [ ] b. Change OR gate to AND gate
- [x] c. Change output AND gate to output OR gate

(a) Multiple emendation choice.



Match the correct option on the right that modifies the below logic circuit to correspond with each of the boolean expressions on the left

f = ((¬a∧¬b)∨(¬(¬c∧d))) Change OR gate to AND gate.
f = ((¬a∨¬b)∨(¬(¬c∨d))) Change input AND gate to input OR gate.
f = ((¬a∨¬b)∧(¬(¬c∧d))) Choose...

Choose...
Change output OR gate to output AND gate.
Change input AND gate to input OR gate.
Change OR gate to AND gate.

(b) Match pairs.

Fig. 8. Generated exercises for Moodle.

Listing 5 shows an eduTest specification with some exercises. Line 1 states that the exercises can be solved in any order. Lines 2–9 define two exercises of kind multiple choice emendation, which allow a single attempt (retry=no) and use the mutants generated by the mtg1 operator. Lines 5–8 specify the statement of the exercises generated from the seed models lc1.model and lc2.model. The *professor* may create these models by hand, or as introduced in Section 3, she can generate them automatically by means of the seed synthesizer provided by the Wodel-Edu tool.

Lines 10–16 define two exercises of the match pairs kind. The %text('bool-exp') variable is automatically substituted by the boolean expression that defines the LC of the corresponding model. Note that the *professor* can use this variable both inline (lines 6 and 8) or as a parameter of the exercises template (line 11), when it is written as text='bool-exp'. This textual model representation is generated by the extension point provided by Wodel-Edu to obtain alternative textual representations from models (see Fig. 3). This approach saves the effort to calculate the boolean expression for each seed model, and is especially beneficial when using the automated synthesis of a high number of seed models. Moreover, it enables defining additional alternative representations, e.g., in domains such as automata, we can both obtain the regular expression or the regular grammar equivalent to the automaton [10].

As an example of the generated exercises, Fig. 8a shows an exercise of the multiple emendation choice kind, where the application of the mtg1 and mtg2 mutation operators of Listing 1 has produced both the text option(s) to emend the diagram and the one(s) applicable to the diagram but leading to different LCs. Fig. 8b shows a match pairs exercise. Both exercises have been generated for the Moodle platform.

5. Impact and evaluation

The creation of exercises – and their grading – is a recurring task of professors nowadays. A tool like Wodel-Edu automates many of the steps of this process. Next we briefly report on a usability evaluation from the point of view of the professor (Section 5.1) and the students (Section 5.2).

5.1. Usability evaluation: professor role

To evaluate Wodel-Edu from the point of view of the *professor*, we designed an experiment where, after a brief introduction, participants had to use Wodel-Edu to generate exercises in the LC domain. Then, we assessed the tool's usability using

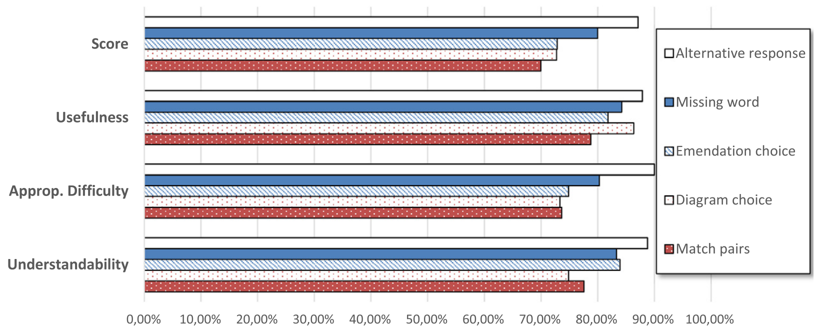


Fig. 9. Exercise quality results for DFA.

a System Usability Scale (SUS) questionnaire [5]. The materials and the raw data of the experiment are available at https://gomezabajo.github.io/Wodel/wodel-edu-eval.html.

The experiment involved 10 participants, all academics, researchers in computer science, and 6 of them professors involved in teaching at the University level. We collected their background using three Likert-scale questions. They declared good knowledge of Eclipse (4.1 out of 5), being knowledgeable about Moodle (3.7 out of 5) and being acquainted with LCs (3.3 out of 5).

Wodel-Edu obtained a SUS score of 74 (out of 100), which, according to [2], can be qualified as *Good* (interval [71.4–85.5]). The questionnaire also included two specific questions on Wodel-Edu, one with a positive tone and one with a negative tone. Specifically, the participants valued the exercises as very useful to learn the subject (4.7 out of 5), and disagreed that they could have defined the same exercises by themselves with less effort (2.2 out of 5, where 1 is “completely disagree”). Finally, we also asked participants to list three positive and negative points about the tool. Regarding the positive aspects, the participants found Wodel-Edu easy to use, appreciated its integration with Eclipse and praised the diversity of the generated exercises. On the contrary, a point for improvement is better feedback to the user when the tool is generating the mutants and the resources of the exercises.

Overall, we can conclude that participants found Wodel-Edu usable and useful, but we also noted some points for improvement. For the experiment, the main threats to validity are the reduced number of participants and the focus on one domain (LCs). In the future, we plan to replicate the experiment on a larger scale.

5.2. *Quality of generated exercises: student role*

We applied our approach in real university courses on automata theory of the computer science degree at the Universidad Autónoma de Madrid. For this, we produced exercises on deterministic finite automata (DFA) and pushdown automata. We evaluated the quality of such exercises with the 68 students enrolled in the course [9]. Fig. 9 summarizes the results for DFA. We generated 10 exercises overall, of 5 different types (we did not evaluate questions of types *text drag and drop* or *multiple text choice*), using 8 mutation operators.

The first series (Score) shows the results obtained by the students with each considered exercise type. Overall, students performed reasonably well, between 70% (for *match pairs*) and 87.12% (for *alternative response*). These results had a slight correlation with the final grade of the students in the course (close to 0.5).

After performing the tests, we asked the students to evaluate the usefulness, appropriate difficulty and understandability of the exercises. Usefulness ranges between 78.8% (for *match pairs*) and 87.9% (for *alternative response*). *Alternative response* exercises were perceived as the easiest to understand (88.8%), while the most difficult ones were those of type *diagram choice* (74.85%). Difficulty was deemed more appropriate for *alternative response* (90%), while the lowest value was for *diagram choice* (73.3%). Overall, we can conclude that these results are very positive, but we aim at performing subsequent evaluations in further domains, like LCs.

A Moodle installation with the exercises produced is available at http://moodle.wodel.eu (username: demo, password: Wodel-Edu4Moodle), and a more detailed analysis of results is available at [9].

6. Related work

Some works exist to automate the generation of exercises, but their scope is limited to a specific domain, like automata [1,15], entity-relationship diagrams [16], cybersecurity [14], English language verb tenses [7], or programming languages [6]. Instead, Wodel-Edu is domain-independent. However, it is restricted to domains in which solutions to exercises can be expressed diagrammatically (e.g., an LC), and whose correctness can be contrasted against a textual specification (e.g., a boolean formula). This means that our approach cannot support open-ended questions or exercise types that provide substantial freedom to the student.

Generally, approaches to exercise generation and automatic grading are either based on modification (like Wodel-Edu) or comparison. Regarding the former, Sadigh *et al.* [15] propose the use of mutation to generate variants of a seed model (a finite automaton), and then SAT solving to create solutions to various problems. The approach is specific to a domain, and rather a proposal without implementation.

Comparison-based approaches achieve automated grading by means of comparison algorithms [1,3,13,16], which may be problematic if the exercise admits multiple solutions. This is not a limitation of Wodel-Edu. If an exercise has several – semantically equivalent – solutions, the student is presented just one of them, and then some mutations of it. The *equivalent detection* extension point (cf. Fig. 3) can be used to check that mutations are not equivalent to the original model, ensuring that mutations are not solutions to the exercise. This situation occurs in our running example, since there are many equivalent circuits for the same boolean formula.

7. Conclusions and future work

Creating and grading on-line exercises is a recurring task within higher education. Wodel-Edu automates this process by using a model-based approach to generate and evaluate diagram-based exercises. It has the advantage of being domain-independent, supporting automated grading, and enabling a large-scale generation of exercises of seven kinds. The tool offers advanced functionality for the automated generation of seed models, and to calculate alternative textual representations of the models. We have used Wodel-Edu in the domains of LCs and automata, showing the versatility and usefulness of this solution.

In the future, we plan to extend Wodel-Edu to support gamification, and to generate other types of exercises, e.g., supporting interactivity via direct diagram manipulation. In addition, another target is to use our approach in other domains, such as software design and electric circuits. Finally, we are considering performing a larger-scale evaluation of Wodel-Edu from the point of view of the professor.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Special gratitude to Andrés Rico-Fernández and Jaime Velázquez Pazos for their help with the Wodel-Edu implementation, building the code generators for the Android and iOS exercises applications, respectively, and to all participants in the evaluation. Project partially funded by the Spanish MICINN (PID2021-122270OB-I00, TED2021-129381B-C21).

References

[1] R. Alur, L. D'Antoni, S. Gulwani, D. Kini, M. Viswanathan, Automated grading of DFA constructions, in: IJCAI, AAAI Press, 2013, pp. 1976–1982.
[2] A. Bangor, P. Kortum, J. Miller, Determining what individual SUS scores mean: adding an adjective rating scale, J. Usability Stud. 4 (2009) 114–123.
[3] W. Bian, O. Alam, J. Kienzle, Is automated grading of models effective? Assessing automated grading of class diagrams, in: MoDELS, ACM, 2020, pp. 365–376.
[4] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, second edition, Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, 2017.
[5] J. Brooke, et al., SUS-a quick and dirty usability scale, in: Usability Evaluation in Industry, 1996, p. 189.
[6] M. Christian, B. Trivedi, A comparison of existing tools for evaluation of programming exercises, in: ICTCS, ACM, 2016.
[7] K. Ferreira, A.R. Pereira Jr., Verb tense classification and automatic exercise generation, in: Brazilian Symposium on Multimedia and the Web, ACM, 2018, pp. 105–108.
[8] P. Gómez-Abajo, E. Guerra, J. de Lara, M.G. Merayo, A tool for domain-independent model mutation, Sci. Comput. Program. 163 (2018) 85–92.
[9] P. Gómez-Abajo, E. Guerra, J. de Lara, Automated generation and correction of diagram-based exercises for Moodle, Under evaluation, 2023.
[10] P. Gómez-Abajo, A. Rico-Fernández, E. Guerra, J. de Lara, Wodel-Edu: an MDE solution for the generation and evaluation of diagram-based exercises, in: ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2021), 2021, pp. 600–605.
[11] M. Kuhlmann, M. Gogolla, From UML and OCL to relational logic and back, in: MoDELS, Springer, 2012, pp. 415–431.
[12] S. Kurbakova, Z. Volkova, A. Kurbakov, Virtual learning and educational environment: new opportunities and challenges under the COVID-19 pandemic, in: ICEMT, ACM, USA, 2020, pp. 167–171.
[13] T. Reischmann, H. Kuchen, A web-based e-assessment tool for design patterns in UML class diagrams, in: SAC, ACM, 2019, pp. 2435–2444.
[14] M. Ribaudo, A. Valenza, Semi-automatic generation of cybersecurity exercises: a preliminary proposal, in: EnSEmble Workshop, ACM, 2019, pp. 16–21.
[15] D. Sadigh, S.A. Seshia, M. Gupta, Automating exercise generation: a step towards meeting the MOOC challenge for embedded systems, in: WESE, ACM, 2013, pp. 2:1–2:8.
[16] P.G. Thomas, N. Smith, K.G. Waugh, Computer assisted assessment of diagrams, SIGCSE Bull. 39 (2007) 68–72.