FlowTGE: Automating Functional Testing of Executable Business Process Models Based on BPMN

Tomás Lopes[1] [0000−0002−5454−2927] and Sérgio Guerreiro[1,2,3] [0000−0002−8627−3338]

[1] Link Consulting SA, Av. Duque de Ávila 23, 1000-138 Lisbon, Portugal
{tomas.lopes,sergio.guerreiro}@linkconsulting.com
[2] INESC-ID, R. Alves Redol 9, 1000-029 Lisbon, Portugal
[3] Instituto Superior Técnico, Av. Rovisco Pais 1, 1049-001 Lisbon, Portugal
sergio.guerreiro@tecnico.ulisboa.pt

Abstract. Testing business process models is vital for guaranteeing the correct operation of processes and ensuring they comply with requirements and regulatory norms. Often performed manually, automating process model testing expedites testing efforts and reduces human error. This paper presents FlowTGE, a tool for end-to-end black/gray-box testing of executable business process models based on the BPMN language. This tool is built on top of the bPERFECT framework and is designed to tackle the slow and error-prone nature of manual process testing in the context of workflow automation systems. FlowTGE is split into two components — the Generator and the Executor — which, respectively, handle the automated generation of test cases from executable BPMN-based process models and the execution of said test cases through process simulations using workflow automation functionalities. Evaluation shows promising results regarding user perceptions of usefulness and ease of use and demonstrates the tool's suitability to be applied to a significant portion of process-testing-related challenges.

Keywords: BPMN · Business process · Test automation · Business process management · Business process testing · Model-based testing

1 Introduction

Business processes lie at the core of organizations [11]. Defined as "a collection of inter-related events, activities, and decision points that involve a number of actors and objects, which collectively lead to an outcome that is of value to at least one customer" [3], the efficiency and effectiveness of business processes directly influence customer experience and the quality of organizational outputs [11].

Business Process Management (BPM) often relies on using process models that represent people's understanding of how work is done, abstracting away unimportant details. A process model describes all the different ways to execute a business process, differing from the notion of a process instance which consists

of one specific execution of a process [3]. Several languages may be used to represent process models, with the most common one being the Business Process Model and Notation (BPMN) language, often regarded as the de facto standard for process modeling.

Technological and organizational growth has led to a massive hike in the complexity of business processes. Maintaining their correctness and compliance with established requirements has become a significant challenge for companies [5]. Business Process Testing (BPT) activities have become essential in guaranteeing that processes continue to bring value as they evolve [5].

Furthermore, the executability of these models has become a critical aspect of BPM, as it determines the extent to which a process can be automated and executed through implementation in BPM systems or other process-aware information systems [3]. BPM systems use process models to guide the flow of work, automatically routing work items to the appropriate people and systems at each step of the process with capabilities for monitoring and process analysis [3].

The main goal of this paper is to discuss the design and implementation of the FlowTGE tool, an automated functional testing solution for assessing BPMN-based executable business process models. Three basic requirements are set: (i) automatically generate test cases from executable process models based on BPMN, (ii) execute process test cases, and (iii) produce easily interpretable test case execution reports.

2 Related Work

A literature review about BPT was recently published, which summarizes and analyzes over 30 studies detailing different approaches and techniques for process testing and verification, focusing on the BPMN language [5]. Many studies reviewed in this paper describe specific testing-related techniques that may be adapted and combined to create a more sophisticated all-encompassing solution:

- [2, 6, 9, 13, 14] describe different ways to transform BPMN models into more elementary graph-like structures which facilitate further analysis, namely path discovery.
- [10, 12–14] use a Depth-First Search (DFS) based path discovery algorithm to extract the existing execution scenarios from a process.
- [8] implement a recursion delimiter that controls the number of times a given sequence flow may be executed in any given execution scenario.
- [8, 14] use a simple tabular format to represent test cases and execution scenarios. These tables contain information such as the corresponding path, pre-conditions, inputs, and post-conditions.

Combining and adapting these techniques shows considerable potential to create new and disruptive BPT solutions. This literature review also proposes a simple classification system for BPT types, as follows [5]:

- Black/gray-box: used to verify basic functional requirements,

– Regression: used to test potentially breaking changes, and
– Integration: used to test specific service-related implementation details

Finally, this review proposes the Business Process Evaluation and Research Framework for Enhancement and Continuous Testing (bPERFECT) framework that aims to guide BPT research, consisting of a series of high-level instructions that all future BPT approaches should follow to facilitate knowledge sharing and boost interoperability and intercompatibility [5]: (i) model the process, (ii) determine constraints, (iii) convert to graph structure, (iv) extract paths, (v) determine test data, (vi) determine testing type, (vii) generate test cases, and (viii) execute test cases.

The classification of testing types, the bPERFECT framework, and the analysis of several existing BPT-related techniques constitute a solid starting point for developing new and innovative solutions that provide added functionality and extend the scope of current BPT approaches.

3 Flow Test Generation and Execution

This section describes the architectural and implementation details of Flow Test Generation and Execution (FlowTGE), an end-to-end black/gray-box business process model testing tool with the capability of automating key parts of the assessment of the correctness of executable process models and providing insights for their improvement.

FlowTGE is designed to automate many steps associated with end-to-end process model testing, including execution scenario determination, test data generation, and test execution. The use of bPERFECT as a reference framework ensures that it is aligned with existing BPT methodologies while also following the best practices in the field.

FlowTGE is split into two components: the Generator and the Executor. Each of these two components is made up of 3-4 core sub-components which are run sequentially to achieve the desired objective.

Table 1 shows the correspondence between bPERFECT steps, specific techniques used in FlowTGE, and the studies from which those techniques were adapted.

3.1 Generator

The Generator is the component of FlowTGE that takes care of everything related to the generation of test cases from process models.

The Generator receives a business process model as input and outputs a set of test cases. Each test case consists of a sequence of detailed steps (such as “Actor A assigns variable X during Task T” or “Actor A executes Task T, meeting constraint C”) which, when followed, lead to the successful termination of the process, per the model.

The Generator comprises four core sub-components – BPMN-to-Graph, Path Extractor, Constraint Extractor, and Test Case Writer. The core sub-components

Table 1. Mapping from bPERFECT steps to specific techniques used in FlowTGE.

bPERFECT steps	FlowTGE techniques	References
1. Model the process	Model using BPMN variant	–
2. Determine constraints	Parse conditions from sequence flows	–
3. Convert to graph	Convert to directed graph	[2, 6, 9, 13, 14]
4. Extract paths	Apply adapted DFS	[8, 10, 12–14]
5. Determine test data	Use constraint solver	[8, 14]
6. Determine testing type	Perform black/gray-box testing	[1, 8, 13, 14]
7. Generate test cases	Write in tabular format	[8, 14]
8. Execute test cases	Simulate using API	–

are run sequentially, each taking as input some of the outputs produced by the sub-components run beforehand.

BPMN-to-Graph The BPMN-to-Graph sub-component, as the name suggests, handles the transformation of the process model into a (directed) graph.

To accomplish this, the model's sequence flows are parsed from the BPMN XML file. Each sequence flow contains a source reference and a target reference which may each be a task, an event, or a gateway. The set of all sequence flows constitutes the edges of the graph, and the union of all source references and all target references constitutes the nodes of the graph. Collapsed subprocess nodes are recursively substituted by the corresponding process graphs.

Path Extractor The Path Extractor sub-component computes all desired paths from the start node to the end node(s) in the graph outputted by the previous sub-component. In this context, a "path" refers to a single end-to-end execution scenario that is valid per the model. Since these may include parallel activities, one path may actually include several parallel sub-paths.

Beginning with the start event node, a DFS is used to recursively explore each node's successors, updating a `visited` dictionary that keeps track of the number of times each node has been visited in the path currently being explored. A parameter controls how many times any specific node may be visited, allowing the exploration of cycles. Upon reaching an end event, the path is added to a list and the search backtracks, exploring paths via other successors until there are no more alternatives.

After the initial path extraction, a parallel sub-path merging operation between paths that contain the same AND-split node is performed which, under the assumption that there are no race conditions between branches that may execute in parallel, sequentializes parallel tasks.

Constraint Extractor The Constraint Extractor sub-component determines which constraints must apply for each path to be followed.

For each path, this sub-component of FlowTGE parses the constraints stored as edge data (sequence flow labels) and stores them in a `TestDatum` object containing a path and a collection of constraints. Each constraint is associated with the task where it must be satisfied to proceed with the test case (see Figure 1).

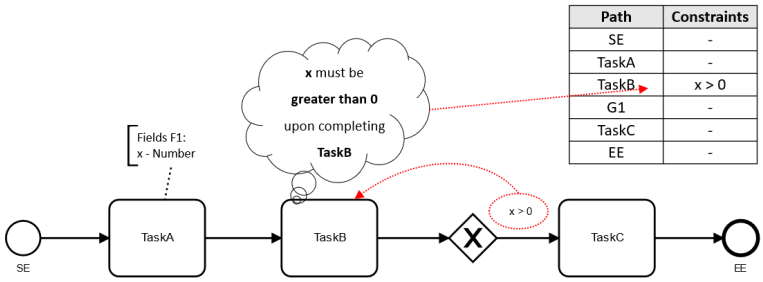

Fig. 1. Extracting constraints from a path.

Test Case Writer Finally, the Test Case Writer generates the test cases based on the acquired information about the possible flows. One test case is generated for each path, representing one possible execution scenario for the process.

Each test case consists of pre-conditions, a sequence of execution steps, and post-conditions. The execution steps are displayed in the test case in tabular form, as done first by [8, 14]. Each step may be of one of three different types: (i) *Instantiate process*, (ii) *Assign variable*, or (iii) *Execute task*.

3.2 Executor

The Executor is the component of FlowTGE that handles the execution of test cases generated from business process models. Before execution, test data is also generated for each test case based on the associated constraints.

This component receives a set of test cases as input and outputs relevant information about the execution of the tests. Test execution data includes aspects such as passing and failed tests, reasons for failure, and execution time.

Similarly to the Generator, the Executor's three core sub-components – Pre-Processor, Solution Mapper, and Test Case Simulator – are run sequentially.

Pre-Processor The Pre-Processor parses the test case specifications and produces additional information about the variables and constraints that may not be explicit in the tests. The tasks performed by this sub-component include variable parsing, constraint parsing, and variable domain estimation.

Solution Mapper The Solution Mapper sub-component determines possible values for the variables involved in each test case that meet all the constraints specified for said test case.

In order to support arbitrarily complex constraints, a constraint solver was used for this task, similar to the approach presented by [4] and proposed by [12]. More specifically, Microsoft's Z3 Theorem Prover[4], an SMT solver (a generalization of classic Boolean satisfiability / SAT) with bindings for programming languages like Python and C#, was used for this purpose. Each test case's constraints (including implicit ones derived by the previous sub-component) are, thus, converted to Z3 Boolean expressions and added to a solving model which, upon calling the `Solve` method, returns one possible value for each variable (`Status.SATISFIABLE`). Alternatively, if the constraints are incompatible, the solver returns `Status.UNSATISFIABLE`; if that happens, the test case is skipped.

Test Case Simulator The Test Case Simulator handles the execution (through simulation) of the test cases and collects data concerning this execution.

The implementation of this execution mechanism differs significantly depending on the workflow engine being used to execute the process models. In the current implementation, edoclink, a document management system implemented, commercialized, and maintained by Link Consulting, S.A. designed around workflow automation, is used for that purpose. As such, the execution of test case steps is achieved using the edoclink Public API. This component may be adapted to support other BPM/workflow automation systems.

If any step of a test case fails to execute, that test case is interrupted and marked as failed. Executing every step of a test case with no errors corresponds to a successful test case.

Users have access to information regarding not just the number of failures but also the reasons for failure. This information, combined with other results and metrics such as execution time, enables the extraction of insights that may be used to correct and enhance the process model.

After executing all test cases, an execution report is generated and presented to the user. This execution report contains all execution data collected for all tests executed, such as (i) a statistical overview of the outcomes of all test cases, (ii) total execution time and execution time per test case, (iii) the values assigned to each variable, and (iv) reasons for failure of each test case (if applicable). The insights extracted from this information facilitate the correction and enhancement of the process model.

4 Evaluation

This section contains all the evaluation procedures carried out to assess end user perceptions of FlowTGE.

4.1 Methodology

To assess the efficiency of the tool, process models with varying amounts of paths were procedurally generated by sequentially chaining cycles. Models were

[4] https://www.microsoft.com/en-us/research/project/z3-3/

generated using this method for each possible value of n from 0 (1 path) to 16 (65536 paths using the `max` mode) and execution times for both components were measured. This experiment showed exponential time and space growth with respect to the number of cycles (as expected by the exponential growth of the number of paths) and linear time and space growth with respect to the number of execution steps, with the Generator being able to generate over 200 execution steps per second.

The main focus of this evaluation, however, consisted of assessing end user perceptions of the tool regarding usefulness and ease of use. On account of this, eight workers at Link Consulting with prior process testing experience were asked to participate in an experiment.

The experiment consisted of users being asked to detect errors in four process models (with deliberately introduced errors) adapted from previous projects by interacting with the Generator to generate the test cases (with the number of paths for each model varying between 5 and 34), interacting with the Executor to execute them, and analyzing the execution reports.

They were then asked to fill out a survey with eight items, each concerning one of three perception-related constructs of Moody's Method Evaluation Model [7]: (i) Perceived Ease of Use, (ii) Perceived Usefulness, and (iii) Intention to Use. A 5-point Likert scale was used to measure each item, reversing the score of negatively worded items (I3, I4, I5, I7) to enable the calculation of numerical measures. Respondents were able to answer each item for the Generator and the Executor independently. Items were shuffled for each respondent, with each one focusing on one of the three perception-related constructs of the model:

I1 I found the tool easy to learn. (Perceived ease of use)
I2 This tool makes it easier to spot errors in process models. (Perceived usefulness)
I3 I found the tool difficult to utilize. (Perceived ease of use)
I4 Overall, I think this tool does not effectively solve the problem of manually testing large process models being time-consuming and error-prone. (Perceived usefulness)
I5 I would not use this tool to test large process models. (Intention to use)
I6 I believe this tool would reduce the effort needed to test large process models. (Perceived usefulness)
I7 The overall procedure for utilizing this tool is complex. (Perceived ease of use)
I8 I plan on using this tool to test processes over manual testing. (Intention to use)

Both components of FlowTGE were deployed in internal environments for end user validation and logs were used to verify user interaction with both of the tool's components.

4.2 Results

The results of the survey carried out concerning user perceptions can be visualized in Figure 2 as box-and-whisker diagrams plotted for each tool/model

construct pair, where the blue whiskers represent the minimum and maximum scores for that item, the edges of the blue boxes represent the first and third quartiles, and the green horizontal bars represent the second quartiles (the medians). Additionally, mean scores were plotted as black dots with black labels.

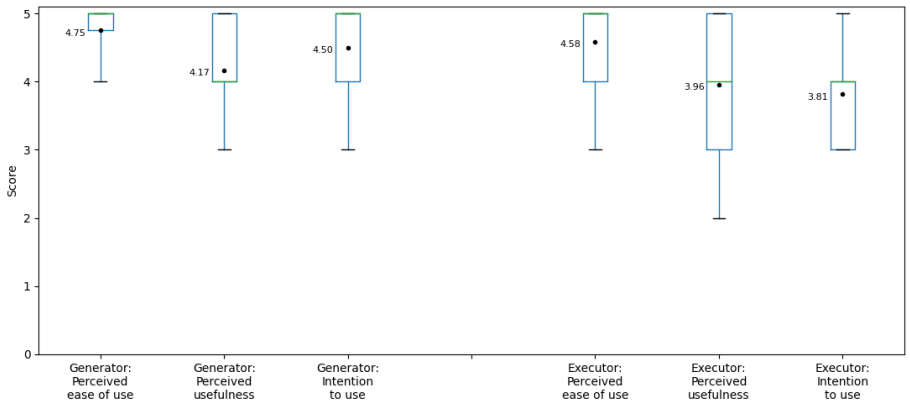

Fig. 2. Box plots of survey scores for each tool/construct pair.

Scores for perceived ease of use were nearly perfect, with both mean scores greater than 4.5 and very little divergence in answers, which likely stems from the lack of effort required from users to use the tool (usage consists of simply pressing one to two buttons and, in the case of the Executor, filling out a simple form, with all of the work being done by the tool in the background).

Scores for items concerning perceived usefulness were still quite positive, albeit taking a noticeable hit when compared to the items concerning perceived ease of use. Furthermore, larger interquartile ranges for most items indicate a larger degree of divergence between respondents. It is also worth noting that scores for the Generator are slightly higher than for the Executor. Hence, despite results still being positive, they show mild concerns regarding the tool's practicality and adequacy at solving the problem at hand.

Finally, as for intention to use, although scores for both the Generator and the Executor can still be considered quite positive, similarly to what happens for perceived usefulness, scores for the Generator are noticeably higher than the Executor's. Respondents showed great intent on continuing to use the Generator, which already brings value independently, while showing some skepticism towards using the Executor, which depends on the Generator to function.

Further conversations with the respondents were carried out with the intent of analyzing why scores for the Executor were lower across the board. Two main limitations regarding the Executor were gathered. Firstly, variables with complex data types are unsupported by the Executor's Solution Mapper, leading to the need for manual assignment of field values or process model refactoring efforts

(removing variables with those data types) to execute test cases for processes containing variables with such data types. Additionally, flexible processes that allow the execution of activities in an ad hoc fashion cannot be comprehensively tested using FlowTGE. Overall, users seemed keen on using the Generator to derive the valid execution scenarios from a model, showing intent to use the Executor to test critical execution scenarios while opting to validate the remaining execution scenarios solely based on the content of the test cases themselves.

The evaluation carried out poses some limitations. Namely, the number of participating users (constrained by the reduced number of employees involved in process testing) is quite small, leading to low statistical power. Furthermore, previous user experience with similar tools designed to facilitate and accelerate process model testing was not measured. Vagueness and possible ambiguity of terms used in the survey items may have also impacted the reliability of the findings.

In any case, evaluation results provide an optimistic outlook on user impressions and intentions, with FlowTGE's functionality successfully fulfilling its objectives while showing great potential for further improvements.

5 Conclusion

This paper presents FlowTGE, a two-component tool that can accelerate and improve the testing of executable BPMN-based process model testing by automating critical tasks typically done manually. Namely, FlowTGE can automatically generate and execute test cases from business process models, thus helping to ensure the proper operation of the processes and reducing manual process testing efforts.

As of writing this paper, the tool is being used internally at Link Consulting and has been integrated with its process management tool suite (Atlas and edoclink). Both components of FlowTGE are already in use today in the context of process management and business consulting projects.

In future work, the tool will be extended to tackle the limitations pinpointed during evaluation, namely its inability to deal with complex data types. Additionally, the constant context switching involved in a typical process build/test loop leaves an opportunity to improve the cohesion of such activities through, for instance, the creation of a common environment for modeling and testing.

Acknowledgment

This work was supported by a pre-registered project, named as eProcess, which is under national funds with reference C669314338-00003137 (LINK CONSULTING – TECNOLOGIAS DE INFORMAÇÃO S.A.).

References

1. Buchs, D., Lucio, L., Chen, A.: Model checking techniques for test generation from business process models. In: Reliable Software Technologies – Ada-Europe

2009: 14th Ada-Europe International Conference on Reliable Software Technologies, Brest, France, June 8-12, 2009, Proceedings. pp. 59–74. Springer, Brest, France (2009). https://doi.org/10.1007/978-3-642-01924-1_5
2. Dechsupa, C., Vatanawood, W., Thongtak, A.: An automated framework for BPMN model verification achieving branch coverage. Engineering Journal-Thailand **25**(2), 135–150 (2021). https://doi.org/10.4186/ej.2021.25.2.135
3. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Berlin, Germany (2018)
4. Jahan, H., Rao, S., Liu, D.: Test case generation for BPEL-based web service composition using colored Petri nets. In: 2016 International Conference on Progress in Informatics and Computing (PIC 2016). pp. 623–628. IEEE, Shanghai, China (2016). https://doi.org/10.1109/PIC.2016.7949575
5. Lopes, T., Guerreiro, S.: Assessing business process models: a literature review on techniques for BPMN testing and formal verification. Business Process Management Journal **29**(8), 133–162 (2023). https://doi.org/10.1108/BPMJ-11-2022-0557
6. Meghzili, S., Chaoui, A., Strecker, M., Kerkouche, E.: An approach for the transformation and verification of BPMN models to colored Petri nets models. International Journal of Software Innovation **8**(1), 17–49 (2020). https://doi.org/10.4018/IJSI.2020010102
7. Moody, D.: The Method Evaluation Model: a theoretical model for validating information systems design methods. In: Proceedings of the 11th European Conference on Information Systems, ECIS 2003, Naples, Italy 16-21 June 2003. Naples, Italy (2003), https://aisel.aisnet.org/ecis2003/79
8. de Moura, J.L., Charão, A.S., Lima, J.C.D., de Oliveira Stein, B.: Test case generation from BPMN models for automated testing of web-based BPM applications. In: 2017 17th International Conference on Computational Science and Its Applications (ICCSA 2017). pp. 1–7. IEEE, Trieste, Italy (2017). https://doi.org/10.1109/ICCSA.2017.7999652
9. Nazaruka, E., Ovchinnikova, V., Alksnis, G., Sukovskis, U.: Verification of BPMN model functional completeness by using the Topological Functioning Model. In: ENASE 2016: Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering. pp. 349–358. SciTePress, Rome, Italy (2016). https://doi.org/10.5220/0005930903490358
10. Paiva, A.C.R., Flores, N.H., Faria, J.P., Marques, J.M.G.: End-to-end automatic business process validation. Procedia Computer Science **130**, 999–1004 (2018). https://doi.org/10.1016/j.procs.2018.04.104
11. Rosemann, M.: Foreword, 2018. In M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Berlin, Germany: Springer, 2018, pp. vii-viii
12. Schneid, K., Stapper, L., Thöne, S., Kuchen, H.: Automated regression tests: a no-code approach for BPMN-based Process-Driven Applications. In: 2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC). pp. 31–40. IEEE, Gold Coast, Australia (2021). https://doi.org/10.1109/EDOC52215.2021.00014
13. Seqerloo, A.Y., Amiri, M.J., Parsa, S., Koupaee, M.: Automatic test cases generation from business process models. Requirements Engineering **24**, 119–132 (2019). https://doi.org/10.1007/s00766-018-0304-3
14. Yotyawilai, P., Suwannasart, T.: Design of a tool for generating test cases from BPMN. In: 2014 International Conference on Data and Software Engineering (ICODSE). pp. 1–6. IEEE, Bandung, Indonesia (2014). https://doi.org/10.1109/ICODSE.2014.7062692