A Survey on Model-Based Testing Tools for Test Case Generation

A Survey on Model-Based Testing Tools for Test Case Generation

Wenbin Li, Franck Le Gall, Naum Spaseski

Easy Global Market, Sophia Antipolis, France
{firstname.lastname}@eglobalmark.com

Abstract. Compared to traditional testing methods, Model-Based Testing (MBT) is able to manage and accomplish testing tasks in a cheaper and more efficient way. A number of MBT tools are developed to support MBT activities in the past few years, whereas the characteristics of these tools largely vary from one to another and users without prior knowledge can hardly choose appropriate tools. This paper aims at providing a survey on the emerging MBT tools following a list of criteria emphasizing on test case generation while illustrating aspects of test data and test script generation. Firstly, we introduce the general MBT process for a common understanding; we then present a list of criteria oriented to test case generation covering fours dimensions i.e., model specification, test generation, test description and overall support; following our proposed criteria, we survey and characterize the emerging MBT tools; at last we summarize the current limitations based on our survey and shed light on further directions of MBT tool development.

Keywords: model-based testing, survey, tool, test case, test generation, model specification, test description

1 Introduction

Model-Based Testing (MBT) designates any kind of "testing based on or involving models" [1]. Models represent the system under test (SUT), its environment, or the test itself, which directly supports test analysis, planning, control, implementation, execution and reporting activities. According to the World Quality Report 2016 [2], the future testing technologies require agile development operations for more intelligence-led testing to meet speed, quality and cost imperatives. MBT, as a promising solution, aims at formalizing and automating as many activities related to testing as possible and thereby to increase both the efficiency and effectiveness of testing.

Following the progress of Model-Based Engineering [3] technologies, MBT has increasingly attracted research attention. A large number of MBT tools are developed to support the practice and utilization of MBT technologies in real cases. MBT tools provide functions that cover three MBT aspects i.e., generation of test cases, generation of test data and generation of test scripts [4], and are used to conduct different kinds of testing such as functional testing, performance testing and usability testing [5].

Nevertheless, the functions of MBT tools largely vary from one to another. Users without prior knowledge can hardly choose appropriate tools corresponding with their testing needs among the wide list, as MBT tools require input in terms of varied models (e.g., UML, PetriNet, BMPN, etc.) and focus on different MBT aspects with different generation strategies for data, test cases and test scripts. Moreover, most of existing MBT tools support mainly automatic test case generation rather than test data generation and test script generation due to two reasons: firstly test case generation requires complicated strategies involving various test selection criteria from MBT models, and the generation results highly reply on the selected criteria and strategies; secondly test case generation brings much more testing benefits, as the main efforts spent on traditional testing lies in manual preparation of test cases. In order to provide users with a common understanding and systematic comparison, this paper aims at reviewing the recent emerging MBT tools focusing on the test case generation aspect while illustrating aspects of test data and test script generation.

MBT tools have been previously reported and compared in several surveys: the first report is presented in [6] in 2002 to illustrate basic principles of test case generation and relevant tools; a later comparison is made in [7] from perspectives of modeling, test case generation and tool extensibility; since more and more MBT tools rely on state-based models, the review in [8] illustrates the state based MBT tools following criteria of test coverage, automation level and test construction. The most recent work [9] presents an overview of MBT tools focusing on requirement-based designs and also illustrates an example by use of the representative tools. Due to the increasing popularity of MBT, the existing MBT tools rapidly evolve and new available tools emerge every year. In this work, we survey the new emerging tools in the past few years with the focus on test case generation. In order to present a more detailed analysis, we propose and apply a list of criteria oriented to test case generation, and surveys the emerging tools that are not included in previous work or not analyzed with test case generation criteria.

The remainder of the paper is organized as follows: section 2 presents the general MBT workflow. Section 3 introduces the survey criteria oriented to test case generation. Section 4 illustrates the MBT tools and section 5 characterizes the tools following our criteria. Based on the survey, section 6 identifies current limitations and shed light on future directions for MBT tool development. Section 7 concludes our contributions.

2 MBT Workflow at a Glance

Model-based testing is an application of model-based design for generating test cases and executing them against SUT for testing purpose. The MBT process can be generally divided into five steps [10] as shown in Fig. 1.

Step 1. Creation of MBT Models. In the first step, users create *MBT models* from *requirement/system specifications*. The specifications define the testing requirements or the aspects to test of SUT (e.g., functions, behaviors and performances). The created MBT models usually represent high-level abstractions of SUT and are described by formal languages or notations (e.g., UML, PetriNet and BMPN). The formats of MBT models depend on the characteristics of SUT (e.g., function driven or data driven system, deterministic or stochastic system) and the required input formats of MBT tools.

Step 2. Generation of Test Cases. The second step automatically generates *abstract test cases* from MBT models when applying the *test selection criteria*. Test selection criteria guide the generation process by indicating the interesting focus to test, such as certain functions of the SUT or the structure of the MBT model (e.g., state coverage, transition coverage and data flow coverage). Applying different criteria to the same MBT model will generate different sets of test cases. Abstract test cases without implementation details of SUT are generated from the MBT models.

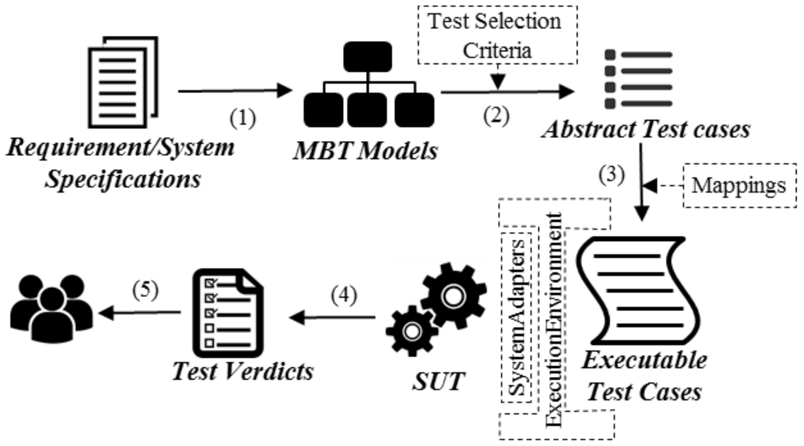

Fig. 1. The MBT workflow

Step 3. Concretization of Test Cases. The third step concretizes the abstract test cases from step 2 to *executable test cases* with the help of *mappings* between the abstraction in MBT models and system implementation details. Executable test cases contain low-level implementation details and can be directly executed against SUT.

Step 4. Execution of Test Cases. The executable test cases are executed against SUT either manually or within an automated test *execution environment*. To automate test execution, *system adapters* are required to provide channels connecting SUT with the test execution environment. During the execution, SUT is respectively stimulated by inputs from each test case, and the reactions of SUT (e.g., output and performance information) are collected to generate *test verdicts*. For each test case, a test verdict is generated indicating if a test passes or fails (or inconclusive).

Step 5. Results Analysis. At the end, testing results are reported to users. For non-satisfactory test verdicts, the MBT process records traces to associate elements from specifications to MBT models and then to test cases, which are used to retrieve possible defects.

Our survey analyzes the support of tools on the five steps with a main focus test case generation from MBT models.

3 Test Case Generation Oriented Criteria

In order to analyze the characteristics and limitations of MBT tools, we propose our survey criteria oriented to test case generation. Our criteria use the taxonomy presented in [11] as a basic reference as it covers the first three MBT steps. In order to focus on the tool support of test case generation, our criteria extend the taxonomy by adding two more dimensions, i.e., test case description and MBT overall support. Our survey criteria are illustrated in Fig. 2.

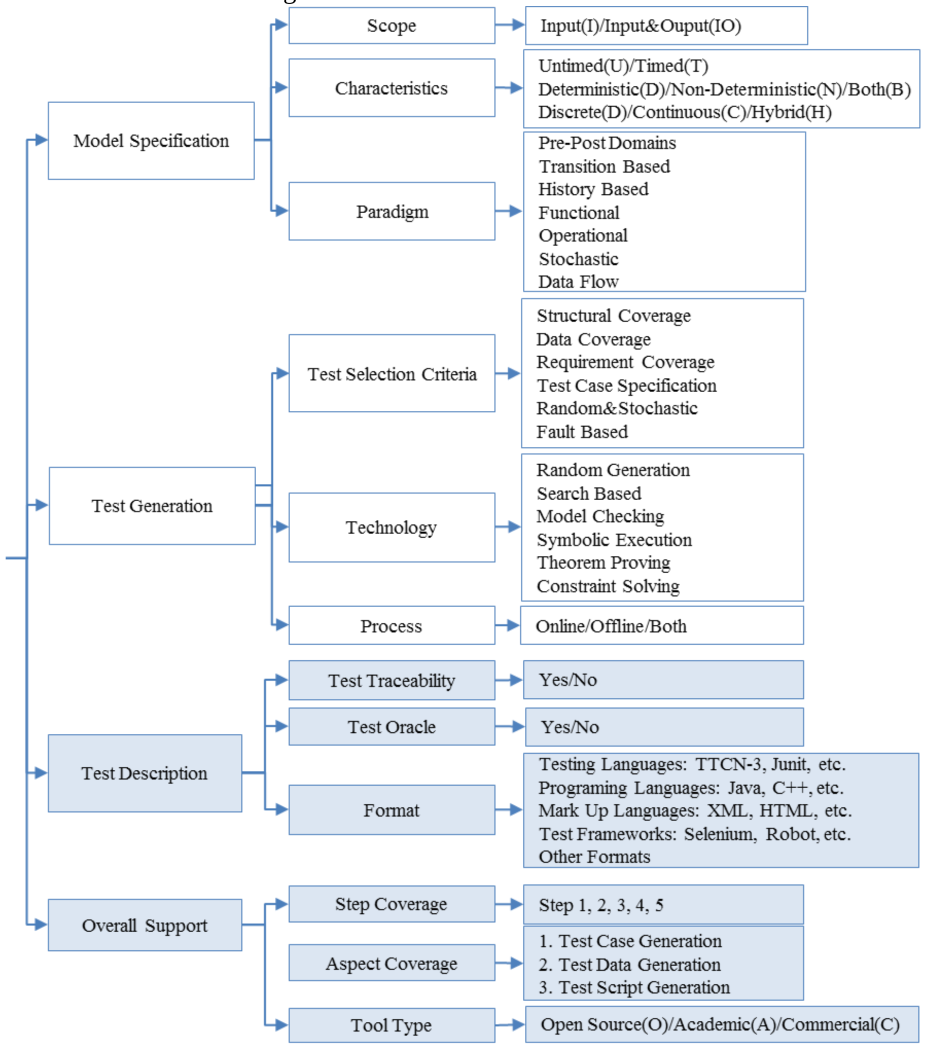

Fig. 2. Survey Criteria oriented to Test Case Generation

The details of the first two dimensions are introduced in [12]. The test description dimension contains the features regarding the contents and formats of the generated test cases:

Test Traceability: This criterion indicates if the generated test cases contain traceability information related to system/requirement specifications or MBT models.

Test Oracle: This criterion indicates if the generated test cases contain the test oracles indicating the pass or fail conditions of test cases.

Formats: This criterion indicates the format of the test cases, which can be categorized into five subtypes: 1) testing languages such as TTCN-3 [11] and JUnit [12]; 2) programming languages such as C++ and Java; 3) markup languages such as XML and HTML; 4) specific formats of test frameworks such as Selenium [13] and Robot [14]. 5) Other self-defined formats or textual descriptions.

The overall support dimension introduces the overall features the MBT tools:

Step Coverage: This criterion indicates which of the five MBT steps introduced in section 2 are supported by the MBT tool. In our paper, partially support is also regarded as support: for instance, a tool offers functionalities of generating execution script is considered to support the test execution step although the tool does not provide an execution environment.

Aspect Coverage: This criterion indicates which of the three MBT aspects, i.e., test case generation, test data generation and test script generation, are supported by the MBT tool.

Tool Type: This criterion indicates the type of the tool, which is open source, academic (that are not open source) or commercial.

Following the criteria, we survey the MBT tools in the following two sections.

4 MBT Tools

In this section, we present the emerging and representative MBT tools for test case generation and discuss their characteristics.

4.1 DIVERSITY

DIVERSITY [15] is an open-source Eclipse based tool for formal analysis. It takes models defined in xLIA (eXecutable Language for Interaction & Assemblage) [15] as input and generates test cases in TTCN-3 which is a standardized testing language for multiple testing purpose support developed by ETSI (European Telecommunication Standards Institute).

Symbolic execution algorithm [16] is used by DIVERSITY to use symbolic values for inputs rather than actual inputs to generate multiple test cases at the same time. Moreover, DIVERSITY provides functionality for validation of MBT models to detect unexpected behaviors such as deadlocks or overdesign of SUT.

4.2 FMBT

FMBT [17] is an open source tool developed by Intel that generates test cases from models written in the AAL/Python pre-postcondition language. FMBT is capable for both online and offline testing on Linux platforms. It provides necessary interfaces to test a wide range of objects from individual C++ classes to GUI applications and distributed systems containing different devices. For now, FMBT supports all MBT steps in commands without graphic interfaces.

4.3 Modbat

Modbat [18] is an open source tool based on extended finite-state machines specialized for testing the APIs of software. A Scala based domain-specific language is used to create models with features for probabilistic and non-deterministic transitions, component models with inheritance, and exceptions. Test cases are generated as sequences of method calls to the API that can be directly executed against SUT.

4.4 TCG

TCG [19] is an open source plugin of LoTuS modeling tool [20] to generate test cases from both probabilistic and non-probabilistic models created by LoTuS. After test generation, TCG is able to proceed a second test selection from the first generation result so as to provide a refined set of test cases. The test generation supports structural model coverage criteria and statistical methods, while the test selection uses five different techniques: test purpose, similarity of paths, weight similarity of paths, post probable path and minimum probability of path.

4.5 Tcases

Tcases [21] is a combinatorial testing tool to test system functions and generate input testing data. An XML document defining SUT as a set of functions is required as input as well as the data space for variables of functions.

The test generation is guided by a predefined data coverage level, by which the number of generated test cases can be controlled. For a set of input variables, Tcases can generate n-wise test cases [22]. The test cases are in stored XML and can be transformed to Junit test cases via integrated convector.

4.6 MISTA

MISTA [23] is an open-source tool that generates test cases from models of finite state machine or function net. Both control-oriented and data-oriented models can be built by MISTA. The formats of test cases cover a number of languages (Java, C, C++, C#, PHP, Python, HTML and VB) and test frameworks (xUnit, Selenium IDE and Robot framework). MISTA supports both online and offline testing.

4.7 MoMuT

MoMuT is a set of model-based test case generation tools that work with UML state machine, timed automata, requirement interfaces and action systems [24]. This tool features a fault-based test case generation strategy [25] that allows mutations to be made on models and generates richer test cases from both original and mutated models to detect if models contain certain user-selectable or seeded faults. A fault localization mechanism is included in MoMuT for debug when a test case fails.

4.8 HTG.

HTG [26] is an academic test case generation tool for hybrid systems. HTG uses hybrid automaton model or SPICE netlists [27] as input and generates test cases in C++. A data coverage measure based on star discrepancy [28] is used to guide the test generation and ensure the test cases are relatively equally distributed over the possible data space. The generated test cases can be applied to numeric simulation and circuit simulation domains.

4.9 Lurette.

Lurette [29] is an automatic test generator for reactive systems. It focuses on environment modeling using Lutin [30] to perform guided random exploration of the SUT environment while taking into account the feedback. For reactive systems, the test cases are realistic input sequences generated from deterministic or non-deterministic environment model. The generation process is online, as their elaboration must be intertwined with the execution of SUT: the SUT output is used as the environment input. The test verdict is automated using Lustre oracles [31].

4.10 VERA

VERA [32] is an academic tool for vulnerability testing, which allows users to define attacker models by means of extended finite state machines, and correspondently generates test cases targeting generic vulnerabilities of Web applications. In order to efficiently perform tests, VERA also provides a library containing common vulnerability test patterns for modelling.

4.11 CompleteTest

CompleteTest [33] is an academic tool for safety intensive critical systems. This tool takes as Function Block Diagram (FBD) as input model and integrates the UPPAAL [34] model checker to perform symbolic reachability analysis on FBD models for test case generation. A set of coverage criteria, including decision coverage and condition coverage are used to guide the generation process. This tools presents a simulation environment to simulate the abstract test cases against FBD models, and also a search-based algorithm to generate executable test cases in C.

4.12 CertifyIt

SmartTesting CertifyIt [35] is a commercial tool for test case generation from models of IBM RSAD [36]. The input models include UML state machine and class diagram, while the generated test cases can be exported in a test environment such as HP quality center and IBM quality manager, or as HTML, XML, Perl/Python Script and Java classes for Junit. In addition, CerfityIt is able to publish the test cases in script format to facilitate test execution, and the traceability is also well maintained for result analysis.

4.13 PragmaDev

PragmaDev Studio [37] is a commercial tool with complete support of all MBT steps. This toolset allows users to create MBT models in SDL and correspondently generates the test cases in TTCN-3. PragmaDev Studio integrates the core of DIVERSITY and uses symbolic execution algorithm for test case generation and MBT model validation. Graphical interfaces are provided for all supported functionalities, and especially, a tracer is designed for testing result analysis to trace elements from requirements, models and test cases via a standard graphical representation. PragmaDev Studio has published a free version for users with small MBT projects.

5 Comparison of MBT tools

In this section, we characterize and compare the MBT tools following our criteria oriented to test case generation, and the results are divided into Table 1 and 2. From the introduction and tables, we can summarize that,

- Current MBT tools does not only support general functional testing for common systems, but also specific testing types such as GUI, API, security and performance testing, and for specific systems such as real time systems and non-deterministic systems.
- The input models of tools are rather rich to capture testing requirements, and most of models are able to include both input and output scope.
- MBT tools offer a wide range of strategies and coverage criteria for test case generation corresponding with different testing requirements.
- Open source and academic tools are available to support MBT in both small and big projects.

Generally, the choice of MBT tools and characteristics to use (e.g., input models, selection criteria and test descriptions) highly vary from one use case to another depending on the testing types (e.g., functional, performance and usability), objects to test (e.g., GUI, API and security), testing environments and so on. How to choose the appropriate tools is not the focus of this paper and a guide can be found in [4].

Table 1. Characterization of reviewed tools - Part 1. Model Specification and Test Description

Tools	Model Specification			Test Description		
	Scope	Charact.	Paradigm	Traceability	Oracles	Formats
DIVERSITY	IO	U/B/D	Transition Based	No	Yes	TTCN-3
FMBT	IO	U/D/D	Pre-Post Domains	No	No	AAL/Python
Modbat	IO	U/B/D	Transition based	Yes	Yes	Other formats
TCG	IO	U/B/D	Transition based	No	No	Other formats
Tcases	I	U/D/D	Pre-Post Domains	No	No	XML, JUnit
MISTA	IO	U/B/D	Operational, Transition Based	No	No	Python, HTML, Selenium, etc.
MoMuT	IO	T/D/H	Transition Based	Yes	No	Other formats
HTG	IO	T/D/H	Transition Based, Functional	No	No	C/C++
Lurette	IO	T/B/H	Functional	Yes	Yes	Lutin
VERA	IO	U/D/D	Transition based	No	Yes	XML
CompleteTest	IO	T/D/H	Data Flow	Yes	Yes	C
CertifyIT	IO	U/D/D	Pre-Post Domains, Transition Based	Yes	Yes	JUnit, HTML, Perl, HP quality center, etc.
Pragmadev	IO	T/B/H	Transition-based, History Based	Yes	Yes	TTCN-3, C/C++

6 Future Directions

Despite the positive aspects, we observe some limitations regarding the tool support on MBT activities that potentially influence the MBT development. In this section, we illustrate future directions for MBT tool development targeting these limitations observed during our survey.

Support of online testing for real time systems. The online testing for real time systems are rarely supported by MBT tools. Timing issues are particularly relevant in the real-time systems, and online testing is generally required as the current system input often relates to the time or the output of another system or environment. Because of the additional degree of freedom, these systems are relatively difficult to test. Online testing for real time systems are expected to be supported by MBT tools in the future.

Generation of test cases in standard testing language. The formats of test cases generated by different tools largely differ from one to another, and combing test cases in different formats into one test suite is difficult. Generation of test cases in standard testing language such as TTCN-3 is thus expected to be supported by more tools. TTCN-3 is a global standardized testing language with multi-purpose support and good test performance. Different execution platforms are available to combine, compile and execute TTCN-3 test cases.

Table 2. Characterization of reviewed tools - Part 2. Test Generation and Overall Support

Tools	Test Generation			Overall Support		
	Test Selection	Technology	Process	Step	Aspect	Type
DIVERSITY	Structural Coverage, Random&Stochastic, Test Case Specification	Symbolic Execution	Offline	1-2	1,2	O
FMBT	Structural Coverage, Test Case Specification	Search Based	Both	1-5	1,3	O
Modbat	Random&Stochastic,	Random Generation, Search Based	Both	1-5	1	O
TCG	Structural Coverage, Random&Stochastic	Search Based, Random Generation	Offline	1-2	1,2	O
Tcases	Data Coverage	Constraint Solving	Offline	1-3	1	O
MISTA	Structural Coverage, Random&Stochastic	Search Based, Random Generation	Both	1-4	1,3	O
MoMuT	Fault Based	Search Based	Offline	1-2	1	A
HTG	Data Coverage	Random Generation, Search Based	Offline	1-3	1	A
Lurette	Random&Stochastic	Random Generation	Online	1-3	1,2	A
VERA	Structural Coverage	Search based	Both	1-5	1,3	A
CompleteTest	Structural Coverage	Model Checking, Search Based	Offline	1-3	1	A
CertifyIT	Structural Coverage, Test Case Specification	Search Based, Model Checking	Offline	1-3	1,3	C
Pragmadev	Structural Coverage, Random&Stochastic	Symbolic Execution	Offline	1-5	1,2	C

Improvement of MBT tool interfaces. User-friendly interfaces are poorly supported by open source and academic tools. Some tools only provide textual interfaces for modeling and command lines for functionalities, without visualization for test generation coverage and execution results. The interfaces of MBT tools can be further improved.

Record of traceability. Traceability is important to maintain the links between test cases and testing requirements and to retrieve possible defects according to execution results. Minority of tools keep record about traceability in test cases and fewer tools support complete traces among testing requirements, models and test cases. The support of traceability by MBT tools can significantly improve the efficiency of result analysis.

Automatic generation of test oracle. Although most of MBT tools use models with both input and output scope, manual update of test cases to add test oracles is often required. Automatic generation of test oracles in test cases requires detailed analysis of MBT models in addition to input and output scope, and test oracle strategies [38] are to implemented into MBT tools.

7 Conclusion

Automatic generation of test cases is one of the main advantages of MBT technology. In the past few years, a large number of MBT tools are developed to meet increasing testing requirements by providing advanced testing solutions. In this paper, we survey the emerging and representative MBT tools for test generation from perspectives of model specification, test generation, test description and overall MBT support. The objective of this work is twofold. On one hand, this paper aims at delivering a systematic analysis and comparison of MBT tools to facilitate the use of MBT and enlarge the MBT community; on the other hand, future directions are illustrated following our survey to attract more research and development attention, to improve the MBT tools and to accelerate the development of MBT activities.

Acknowledgement. Preparation of this paper was supported by the PHANTOM project, which receives funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 688146. The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in this article lies entirely with the author(s).

References

1. Kramer, A., Legeard, B.: Model-based testing essentials: guide to the ISTQB certified model-based tester foundation level. John Wiley & Sons Inc, Hoboken, New Jersey (2016).
2. Capgemini, World Quality Report 2016-17
3. Model-Based Engineering Forum, http://modelbasedengineering.com/
4. Utting, M., Legeard, B.: Practical model-based testing: a tools approach. Morgan Kaufmann Publishers, Amsterdam ; Boston (2007).
5. Spillner, A., Linz, T., Schaefer, H.: Software testing foundations: a study guide for the certified tester exam: foundation level, ISTQB compliant. Rocky Nook, Inc, Santa Barbara, CA (2014).
6. Hartman, A.: Model based test generation tools. Agedis Consortium, URL: http://www. agedis. de/documents/ModelBasedTestGenerationTools_cs. pdf. (2002).
7. Budnik, C.J., Subramanyan, R., Vieira, M.: Peer-to-Peer Comparison of Model-Based Test Tools. GI Jahrestagung (1). 133, 223–226 (2008).
8. Shafique, M., Labiche, Y.: A systematic review of model based testing tool support. Carleton University, Canada, Tech. Rep. Technical Report SCE-10-04. (2010).

9. Marinescu, R., Seceleanu, C., Le Guen, H., Pettersson, P.: A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs. In: Advances in Computers. pp. 89–140. Elsevier (2015).
10. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability. 22, 297–312 (2012).
11. Willcock, C. ed: An introduction to TTCN-3. Wiley, Chichester ; Hoboken, N.J (2011).
12. JUnit, http://junit.org/junit4/
13. Selenium - Web Browser Automation, http://www.seleniumhq.org/
14. Robot Framework, http://robotframework.org/
15. Eclipse Formal Modeling Project, https://projects.eclipse.org/proposals/eclipse-formal-modeling-project
16. Faivre, A., Gaston, C., Gall, P.L.: Symbolic Model Based Testing for Component Oriented Systems. In: Testing of Software and Communicating Systems. pp. 90–106. Springer, Berlin, Heidelberg (2007).
17. FMBT, https://01.org/fmbt/
18. Modbat, https://people.kth.se/~artho/modbat/
19. Muniz, L.L., Netto, U.S., Maia, P.H.M.: TCG: A Model-based Testing Tool for Functional and Statistical Testing. In: ICEIS 2015-Proceedings of the 17th International Conference on Enterprise Information Systems. pp. 404–411 (2015).
20. LoTuS, http://jeri.larces.uece.br/lotus/
21. Tcases, https://github.com/Cornutum/tcases
22. Sanchez, J.: A Review of Pair-wise Testing. arXiv:1606.00288 [cs]. (2016).
23. MISTA - Model-Based Testing, http://cs.boisestate.edu/~dxu/research/MBT.html
24. Momut, https://momut.org/
25. Herzner, W., Schlick, R., Austrian, A., Br, H., Wiessalla, J., Aachen, F.F.: Towards Fault-based Generation of Test Cases for Dependable Embedded Software.
26. HTG, https://sites.google.com/site/htgtestgenerationtool/home
27. Netlist Syntax, http://fides.fe.uni-lj.si/spice/netlist.html
28. LaValle, S.M., Branicky, M.S.: On the Relationship between Classical Grid Search and Probabilistic Roadmaps. 59–75 (2004).
29. Lurrette, http://www-verimag.imag.fr/Lurette,107.html
30. Raymond, P., Roux, Y., Jahier, E.: Lutin: A Language for Specifying and Executing Reactive Scenarios. EURASIP Journal on Embedded Systems. 2008, 753821 (2008).
31. Lustre V6, http://www-verimag.imag.fr/Lustre-V6.html
32. Vera, http://www.spacios.eu/index.php/spacios-tool/
33. CompleteTest, http://www.completetest.org/about/
34. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. (1997).
35. Smartesting CertifyIt, http://www.smartesting.com/en/certifyit/
36. IBM - Rational Software Architect Designer, http://www-03.ibm.com/software/products/en/ratsadesigner
37. PragmaDev - Modeling and Testing tools, http://pragmadev.com/
38. Li, N., Offutt, J.: An Empirical Analysis of Test Oracle Strategies for Model-Based Testing. Presented at the March (2014).