Survey on Automated Software Test Data Generation in Model-Based Testing

Mr. Sumit Bhattacharjee,
Research Scholar, Singhania University,
Rajasthan, India

Dr. Praveen Kumar,
Singhania University, Rajasthan, India

Abstract:*Model-based testing (MBT) is a light-weight formal method which uses models of software systems for the derivation of test suites. In contrast to traditional formal methods, which aim at verifying programs against formal models, MBT aims at gathering insights in the correctness of a program using often incomplete test approaches. The technology gained relevance in practice since around the beginning of 2000. At the point of this writing, in 2011, significant industry applications exist and commercial grade tools are available, as well as many articles are submitted to conferences and workshops. The area is diverse and difficult to navigate. This section attempts to give a survey of the foundations, tools, and applications of MBT. Its goal is to provide the reader with inspiration to read further. Focus is put on behavioral, sometimes also called functional, black-box testing, which tests a program w.r.t. its observable input/output behavior. While there are many other approaches which can be called MBT (stochastic, structural /architectural, white-box, etc.), including them is out of scope for this survey. Historical context is tried to be preserved: even if newer work exists, we try to cite the older one first.*

1. Introduction to Model-Based Testing

One can identify three main schools in MBT: axiomatic approaches, finite state machine (FSM) approaches, and labeled transition system (LTS) approaches. Before digging deeper into those, some general independent notions are introduced. In behavioral MBT, the system-under-test (SUT) is given as a black box which accepts inputs and produces outputs. The SUT has an internal state which changes as it processes inputs and produces output. The model describes possible input/output sequences on a chosen level of abstraction, and is linked to the implementation by a conformance relation. A test selection algorithm derives test cases from the model by choosing a finite subset from the potentially infinite set of sequences specified by the model, using a testing criterion based on a test hypothesisjustifying the adequateness of the selection. Test selection may happen by generating test suites in a suitable language ahead of test execution time, called offline test selection, or maybe intervened with test execution, called online test selection.

1.1. Axiomatic Approaches

Axiomatic foundations of MBT are based on some form of logic calculus. Gaudel summarizes some of the earliest work going back to the 70ties in her seminal paper from 1995 (Gaudel, 1995). Gaudel's paper also gives a framework for MBT based on algebraic specification (see e.g. (Ehrig and Mahr, 1985)), resulting from a decade of work in the area, dating back as early as 1986 (Boug´e et al., 1986). In summary, given a conditional equation like

p(x)! f (g(x); a) = h(x), where f , g, and h are functions of the SUT, a is a constant, p a specified predicate, and x a variable, the objective is to find assignments to x such that the given equality is sufficiently tested.Gaudel et.al developed various notions of test hypotheses, notably regularity and uniformity. Under the regularity hypothesis all possible values for x up to certain complexity n are consideredto provide sufficient test coverage. Under the uniformity hypothesis, a single value per class of input is considered sufficient. In (Boug´e et al., 1986) the authors use logic programming techniques (see e.g. (Sterling and Shapiro, 1994)) to find those values. In the essence, p is broken down into its disjunctive normal form (DNF), where each member represents an atom indicating a value for x. The algebraic approach can only test a single function or sequence, and in its pure form it is not of practical relevance today; however, it was groundbreaking at its time, and the DNF method of test selection is mixed into various more recent approaches.

Dick and Faivre introduced in (1993) a test selection method for VDM (Plat and Larsen, 1992) models based on pre- and post- conditions. The basic idea of using the DNF for deriving inputs is extended for generating sequences as follows. A state machine is constructed where each state represents one of the conjunctions of the DNF of pre- and post conditions of the functions of the model. A transition is drawn between two states S 1 and S 2, labeled with a function call f (x), if there exists an x such that S 1 $\Rightarrow$ pre(f (x)) and post(f (x)) $\Rightarrow$S 2. On this state machine, test selection techniques can be applied as described in Sect. 4.1.2. A theorem proved is required to prove the above implications, which is of course un-decidable for non-trivial domains of x, but heuristics can be used as well to achieve practical results.

The approach of Dick and Faivre was applied and extended in (Helke et al., 1997). Also, (Legeard et al., 2002) is related to it, as well as (Kuliamin et al., 2003). Today, the work of Wolff et.al (2012) is closest to it. These authors use the higher-order logic theorem prover system Isabelle (Paulson, 1994) in which they encoded formalisms for modeling stateless and state-based systems, and implemented numerous test derivation strategies based on according test hypotheses. In this approach, the test hypotheses are an explicit part of the proof obligations, leading to a framework in which interactive theorem proving and testing are seamlessly combined.

Models which use pre- and post conditions can be also instrumented for MBT using random input generation, filtered via the pre-condition, instead of using some form of theorem proving/constraint resolution. Tools like QuickCheck (Claessen and Hughes, 2000) and others are used successfully in practice applying this approach.

1.2. FSM Approaches

The finite state machine (FSM) approach to MBT was initially driven by problems arising in functional testing of hardware circuits. The theory has later been adapted to the context of communication protocols, where FSMs have been used for a long time to reason about behavior. A survey of the area has been given by Lee et.al in (1996).

In the FSM approach the model is formalized by a Mealy machine, where inputs and outputs are paired on each transition.Test selection derives sequences from that machine using some coverage criteria. Most FSM approaches only deal

with deterministic FSMs, which is considered a restriction if one has todeal with reactive or under-specified systems.

Test selection from FSMs has been extensively researched. One of the subjects of this work is discovering assumptions on the model or SUT which would make the testing exhaustive.

Even though equivalence between two given FSMs is decidable, the SUT, considered itself as an FSM, is an 'unknown' black-box, only exposed by its I/O behavior. One can easily see that whatever FSM the tester supposes the SUT to be, in the next step it can behave different. However, completeness can be achieved if the number of states of the SUT FSM has a known maximum, as Chow has shown in (1978) (see also (Vasilevskii, 1973; Lee and Yannakakis, 1996)). A lot of work in the 80s and 90s is about optimizing the number of tests regards length, overlap, and other goals, resulting for example in the Transition-Tour method (Naito and Tsunoyama, 1981) or the Unique-Input-Output method.

Various refinements have been proposed for the FSM approach, and problems have been studied based on it. Huo and Petrenko investigated the implications of the SUT using queues for buffering inputs and outputs (Huo and Petrenko, 2005). This work is important for practical applications as the assumption of input and output enabledness, which assumes that the SUTmachine can accept every input at any time (resp. the model machine /test case every output), is not realistic in real-world test setups. Hierons and Ural (2008); Hierons (2010) have investigateddistributed testing. Hierons (2010) showed that when testing from an FSM it is un-decidable whether there is a strategy for each local tester that is guaranteed to force the SUT into a particular model state.

FSMs are not expressive enough to model real software systems. Therefore, most practical approaches use extended finite state machines (EFSM). Those augment the control state of anFSM with data state variables and parameters), and an update on the state variables which is performed when the rule is taken, which happens if the guard evaluates to true in a given state. In practice, many people just say 'state machine' when in fact they mean an EFSM. A typical instance of an EFSM is a statechart. In a proper EFSM, the domains from which data is drawn are finite, and therefore the EFSM can be unfolded into an FSM, making the foundationsof FSMs available for EFSMs. However, this has its practical limitations, as the size of the expanded FSM can be easily astronomic. A different approach than unfolding is using symboliccomputation on the data domains, applying constraint resolution or theorem proving techniques.

1.3. LTS Approaches

Labeled transition systems (LTS) are a common formalism for describing the operational semantics of process algebra. They have also been used for the foundations of MBT. An earlyannotated bibliography is found in (Brinksma and Tretmans, 2000).

Tretmans described IOCO in (1996) and consolidated the theory in (Tretmans, 2008). IOCO stands for input/output conformance, and defines a relation which describes conformanceof a SUT w.r.t. a model. Tretmans starts from traditional LTS systems which consist of a set of states, a set of labels, a transition relation, and an initial state. He partitions the labels into inputs, outputs, and a symbol for quiescence, a special output. The approach assumes the SUT to be an input enabled LTS which accepts every input in every state. If the

SUT is not naturally input enabled, it is usually made so by wrapping it in a test adapter. Quiescence on the SUT represents the situation that the system is waiting for input, not producing any output by its own, which is in practice often implemented observing timeouts.

As well known, an LTS spawns traces (sequence of labels).Because of non-determinism in the LTS, the same trace may lead to different states in the LTS. The IOCO relation essentially states that a SUT conforms to the model if for every suspension trace of the model, the union of the outputs of all reached states is a superset of the union of the outputs in the according trace of the SUT. Hereby, a suspension trace is a trace which may, in addition to regular labels, contain the quiescence label. Thus the model has 'foreseen' all the outputs the SUT can produce. IOCO is capable of describing internal non-determinism in model and SUT, which distinguishes it from most other approaches. There are numerous extensions ofIOCO, among those real-time extensions (Nielsen and Skou, 2003; Larsen et al., 2004) and extensions for symbolic LTS (Frantzen et al., 2004; Jeannet et al., 2005) using parameterizedlabels and guards. The effect of distributed systems has been investigated for IOCO in (Hierons et al., 2008).

2. Modeling Notations

A variety of notations are in use for describing models for MBT. Notations can be generally partitioned into scenariooriented, state-oriented, and process-oriented. Whether theyare textual or graphical (as in UML) is a cross-cut concern to this. A recent standard produced by ETSI, to which various tool providers contributed, collected a number of tool-independentgeneral requirements on notations for MBT following this taxonomy (ETS, 2011b).

2.1. Scenario-Oriented Notations

Scenario-oriented notations, also called interaction-oriented notations, directly describe input/output sequences between the SUT and its environment as they are visible from the viewpoint of an outside observer ('gods view'). They are most commonly based on some variation of message sequence chart, activity chart (flow chart) (Hartmann et al., 2005;Wieczorek and Stefanescu, 2010), or use case diagram (Kaplan et al., 2008), though textual variations have also been proposed (Grieskamp et al., 2004; Katara and Kervinen, 2006).

Test selection from scenario-based notations is generally simpler than from the other notational styles, because by nature the scenario is already close to a test case. However, scenarios may still need processing for test selection, as input parameters need to be concretized, and choices and loops need expansion. Most existing tools use special approaches and not any of the axiomatic, FSM, or LTS based ones. However, in (Grieskamp et al., 2006a) it is shown how scenarios can be indeed broken down to an LTS-based framework in which they behave similaras other input notations and are amenable for model composition.

2.2. State-Oriented Notations

State-oriented notations describe the SUT by its reaction on an input or output in a given state. As a result the models state is evolved and, in case of Mealy machine approaches, an output maybe produced. State-oriented notations can be given in diagrammatic form (typically, statecharts) or in textual form(guarded update rules in a

programming language, or pre/post conditions on inputs, outputs and model state). They can be mapped to axiomatic, FSM, or LTS based approaches, and can describe deterministic or non-deterministic SUTs.

2.3. Process-Oriented Notations

Process-oriented notations describe the SUT in a procedural style, where inputs and outputs are received and sent as messages on communication channels. Process-algebraic languageslike LOTOS are in use (Tretmans and Brinksma, 2003),as well as programming languages which embed communication channel primitives. Process-oriented notations naturally map to the LTS approach.

3. Tools to be used

There are many MBT tools around; some of them result of research experiments, some of them used internally by an enterprise and not available to the public, and others which are commercially available. Hartman gave an early overview in 2002 (Hartman, 2002), and Legeard and Utting included one in their book from 2007 (Utting and Legeard, 2007). Sincethen, the landscape has changed, and new tools are on the market, whereas others are not longer actively developed. It would be impossible to capture the entire market given more than a short-lived temporary snapshot. Here, three commercial grade tools are sketched which are each on the market for nearly ten years and are actively developed. The reader is encouraged to do own research on tools to evaluate which fit for a given application; the selection given here is not fully representative. For an alternative to commercial tools, one might also check out Binder's recent overview (Binder, 2012) of open-source or open-binary tools.

3.1. Conformiq Designer

The Conformiq Designer7 (Huima, 2007) (formerly called QTronic) has been around since 2006. Developed originally for the purpose of protocol testing, the tool can be used to modela variety of systems. It is based on UML statecharts as a modeling notation, with Java as the action language. Models can also be written entirely in Java. The tool supports compositionof models from multiple components, and timeouts for dealing with real-time. It does not support non-determinism.

Conformiq Designer has its own internal foundational approach, which is probably closest to LTS. A symbolic exploration algorithm is at the heart of the test selection procedure. The tool can be fed with a desired coverage goal (in terms of requirements, diagram structure, or others) and will continue exploration until this goal is reached. Requirements are annotated in the model and represent execution points or transitions which have been reached.

Conformiq Designer can generate test suites in various formats, including common programming languages, TTCN-3, and manual test instructions. The generated test cases can be previewed as message sequence charts by the tool.

Conformiq Designer has so far been used in industrial projects in telecommunication, enterprise IT, automotive, industrial automation, banking, defense and medical application domains.

3.2. SmartestingCertifyIt

The SmartestingCertifyIt tool8 (Legeard and Utting, 2010) (formerly called Smartesting Test Designer) is around since 2002. Coming out of Legeard's, Utting's, and others work around testing from B specifications (Abrial, 1996), the current instance of the tool is based on UML

statecharts, but also supports BPMN scenario-oriented models, and pre/post-condition style models using UML's constraint language OCL.

SmartestingCertifyIt uses a combination of constraint solving, proof and symbolic execution technologies for test generation. Test selection can be based on numerous criteria, including requirements coverage and structural coverage like transition coverage. The tools also supports test selection based on scenarios (in BPMN), similar as Spec Explorer does. The tool generates test suites for offline testing in numerous industry standard formats, and supports traceability back to the model. Non-determinism is not supported.

CertifyIt is dedicated to IT applications, secure electronic transactions and packaged applications such as SAP or Oracle E-Business Suite.

3.3. Spec Explorer

Microsoft Spec Explorer is around since 2002. The current major version, called Spec Explorer 20109, is the third incarnation of this tool family. Developed in 2006 and described in (Grieskamp, 2006; Grieskamp et al., 2011b) it should not be confused with the older version which is described in (Veanes et al., 2008). Spec Explorer was developed at Microsoft Research,which makes it in contrast to the other commercial tools highly documented via research papers, and moved in 2007 into a production environment mainly for its application in Microsoft's Protocol Documentation Program. The tool is integrated into Visual Studio and shipped as a free extension for VS.

The tool is intentionally language agnostic but based on the .Net framework. However, the main notations used for modeling are a combination of guarded-update rules written in C# and scenarios written in a language called Cord (Grieskamp and Kicillof, 2006). The tool supports annotation of requirements, and (via Cord) ways for composing models. Composing a statebased model written in C# with a scenario expressing a test purpose defines a slice of the potentially infinite state model, and is one of the ways how engineers can influence test selection.

The underlying approach of Spec Explorer are interface automata (IA), thus it is an LTS approach supporting (external) non-determinism. Spec Explorer uses a symbolic explorationengine (Grieskamp et al., 2006b) which postpones expansion of parameters until the end of rule execution, allowing selecting parameters dependent on path conditions. The tool supports onlineand offline testing, with offline testing generating C# unit tests. Offline test selection is split into two phases: first the model is mapped into a finite IA, and then traversal techniques are run on that IA to achieve a form of transition coverage.

Spec Explorer has been applied, amongst various internal Microsoft projects, in arguably the largest industry application for MBT up to now, a 350 person year project to test the Microsoft protocol documentation against the protocol implementations (Grieskamp et al., 2011b). In course of this project, the efficiency of MBT could be systematically compared to traditional test automation, measuring an improvement of around40%, in terms of the effort of testing a requirement end-to-end (i.e. from the initial test planning to test execution). Details are found in (Grieskamp et al., 2011b).

4. Conclusion on Model-Based Testing

At the ETSI MBT user conference in Berlin in October 2011[10], over 100 participants from 40 different companies

came together, discussing application experience and tool support.

Many of the academic conferences where general test automation work is published (like ICST, ICTSS (formerly TestCom/FATES), ASE, ISSTA, ISSRE, AST, etc.) regularly see a significant share of papers around MBT. Two Dagstuhl seminars have been conducted around the subject since 2004 (Brinksma et al., 2005; Grieskamp et al., 2011a); the report from the last event lists some of the open problems in thearea. These all document a lively research community and very promising application area.

References:

[1] Gaudel, M. C., 1995. Testing can be formal, too. In: Mosses, P. D., Nielsen, M., Schwartzbach, M.I. (Eds.), TAPSOFT'95: Proc. of the 6th International Joint Conference on Theory and Practice of Software Development, LNCS 915, pp. 82–96. Springer.

[2] Ehrig, H., Mahr, B., 1985. Fundamentals of Algebraic Specification 1: Equations and Initial Semantics. volume 6 of Monographs in Theoretical Computer Science. An EATCS Series.Springer.

[3] Claessen, K., Hughes, J., 2000. QuickCheck: a lightweight tool for random testing of haskell programs. In: Odersky, M., Wadler, P. (Eds.), Proc. of the 5th ACM SIGPLAN International Conference on Functional Programming (ICFP'00), ACM. pp. 268–279.

[4] Helke, S., Neustupny, T., Santen, T., 1997. Automating test case generation from Z specifications with Isabelle. In: Bowen, J.P., Hinchey, M.G., Till, D. (Eds.), Proc. of the 10th International Conference of Z Users on The Z Formal Specification Notation (ZUM'97), pp. 52–71.Springer.

[5] Legeard, B., Peureux, F., Utting, M., 2002. Automated boundary testing from Z and B. In: Eriksson, L. H., Lindsay, P. A. (Eds.), FME 2002:Formal methods-Getting IT Right, LNCS 2391, pp. 21–40. Springer.

[6] Kuliamin, V. V., Petrenko, A. K., Kossatchev, A., Burdonov, I. B., 2003.The UniTesK approach to designing test suites. Programming and Computer Software 29, 310–322.

[7] Vasilevskii, M., 1973. Failure diagnosis of automata.Kibernetika , 98–108.

[8] Lee, D., Yannakakis, M., 1996. Principles and methods of testing finite state machines - a survey.In: Proceedings of the IEEE 84(8), 1090–1123. IEEE Computer Society Press.

[9] Naito, S., Tsunoyama, M., 1981. Fault detection for sequential machines by transition-tours. In: Proc. of the 11th annual International Symposium on Fault-Tolerant Computing (FTCS'81), pp. 238–243.

[10] Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F., Vacelet, N., Utting, M., 2007. A subset of precise UML for model-based testing, in: A-MOST, ACM. pp. 95–104.

[11] Huo, J., Petrenko, A., 2005. Covering transitions of concurrent systems through queues. In: Proc. of the 16th IEEE International Symposium on Software Reliability Engineering, (ISSRE'05), pp. 335–345.

[12] Hierons, R. M., 2010. Reaching and distinguishing states of distributed systems. SIAM J. Comput. 39, 3480–3500.

[13] Nachmanson, L., Veanes, M., Schulte, W., Tillmann, N., Grieskamp, W., 2004. Optimal strategies for testing nondeterministic systems. In: Avrunin, G. S., Rothermel, G. (Eds.), Proc. of the 2004 [14] ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'04), pp. 55–64.

[15] Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., Nachmanson, L., 2008. Model-based testing of object-oriented reactive systems with Spec Explorer. In: Hierons, R. M., Bowen, J. P., Harman, M. (Eds.), Formal Methods and Testing, pp. 39–76. Springer.

[16] Utting, M., Legeard, B., 2007. Practical Model-Based Testing - A Tools Approach. Morgan Kaufmann.

[17] Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F., Vacelet, N., Utting, M., 2007. A subset of precise UML for model-based testing, in: A-MOST, ACM. pp. 95–104.

[18] Grieskamp, W., 2006. Multi-paradigmatic model-based testing. In: (Havelund et al., 2006), pp. 1–19.

[19] Grieskamp, W., Tillmann, N., Veanes, M., 2004. Instrumenting scenarios in a model-driven development environment. Information & Software Technology 46, 1027–1036.