UniTESK: Component Model Based Testing

Alexander K. Petrenko[1], Victor Kuliamin[1] and Andrey Maksimov[1]

[1] Institute for System Programming of Russian Academy of Sciences

(petrenko, kuliamin, andrew)@ispras.ru

Abstract. UniTESK is a testing technology based on formal models or formal specifications of requirements to the behavior of software and hardware components. The most significant applications of UniTESK in industrial projects are described, the experience is summarized, and the prospective directions to the Component Model Based Testing development are estimated.

Keywords. Specification, verification, model-based testing, automated test generation

Key terms. SoftwareSystem, FormalMethod, SpecificationProcess, VerificationProcess

1 Introduction

Model Based testing (MBT) is a rapidly developing domain of software engineering. One of the reasons for such rapid development is the fact that MBT is at the intersection of various other domains of software engineering. In particular, those domains include methods for defining, formalization and modeling of requirements, methods for analysis of both formal specification and formal models as well as the software code, methods for abstraction level control, model transformation and many other software engineering domains. It provides MBT with the ability to quickly adopt the recent achievements proved to be useful in joint domains, in particular, in methods of static analysis and mixed static/dynamic analysis. However, there is no market-ready well-established MBT tool that could be recommended for use in wide range of software development and testing projects. To further develop MBT, we should first analyze the experience gained in the last 15-20 years in this domain. This should help to identify some common problems and focus on their solutions. In this paper, we briefly describe the stages of UniTESK (Unified TEsting & Specification toolKit) development – one of the first MBT tools targeted at testing of wide class of software systems. In the course of the paper, we highlight both positive and negative lessons learned during development and using UniTESK tools. The paper has a subtitle – industrial paper. It means that here we don't reveal any new solutions and don't set any new scientific problems. We analyze the experience and try to learn lessons that would be useful to researchers working in this domain.

The UniTESK technology [1, 2] was initially developed on the basis of the experience gained in the project on the creation of the automated testing KVEST [3] system, which was developed for testing of the real-time operating system kernel. The work started in 1994 when the term Model Based Testing did not exist. This term appeared at the edge of the 21st century. Currently, MBT is rapidly developed. There are many enthusiasts of this approach and many interpretations of the term itself. To properly position UniTESK in the wide spectrum of MBT solutions, we should first clarify the meaning of MBT within the UniTESK framework.

The following definition is currently given in Wikipedia: “Model-based testing is application of Model based design for designing and optionally also executing artifacts to perform software testing. Models can be used to represent the desired behavior of a System Under Test (SUT), or to represent testing strategies and a test environment”.

This definition includes almost all known interpretations of this term. However, most researchers and practitioners mean more specific approaches and testing techniques by MBT. The first main dividing line is the choice of the modeled object: some model the behavior of the target system (SUT), others model the environment of the target system, in particular, the test itself or the testing system, which, of course, are external to the target system. In UniTESK, the model specifies the system behavior. There are also various types of MBT in this approach, which differ in the way of the behavior description. About the first type Jan Paleska [4] says: “the behavior of the system under test (SUT) is specified by a model elaborated in the same style as a model serving for development purposes”. Such specifications or models are called *executable*. The role of the executable model can be played either by prototype implementation algorithm or by some model which explicitly contains the notion of calculation/execution, for example, finite-state machine, Petri net, ASM [5], etc. Examples of other types of MBT, i.e. “*nonexecutable*”, are algebraic specifications, software contracts in the form of pre- and post-conditions of functions. Each of the model types has its own advantages and drawbacks when testing different SUTs. Besides, when generating tests, not only test data and the sequence of calls of the tested functions should be generated. Also required are the “artifacts” mentioned in the Wikipedia definition, for example, test oracles – the components of the test suite that automatically evaluate the results of the SUT execution whether they meet the requirements or not. Executable models often do not allow test oracle creation, but they are very good for generation of test sequences. Software contracts simplify generation of test oracles, but they do not allow effective generation of test sequences. In other words, several types of models required for generation of effective test suites. Back to UniTESK, we can say that the main model type in it is the software contract in the form of pre- and post-conditions of the functions. In addition, online construction of finite-state machine is used in UniTESK making possible the generation of rather non-trivial test sequences.

UniTESK is a technology that can be implemented on various software platforms and, therefore, can be used for testing of API in various programming languages.

Currently, the most actively used implementations of UniTESK are for C, C++, Java, Python. The corresponding tools are: CTESK, C++TESK, JavaTESK, and PyTESK [6].

UniTESK is an academic product developed in ISPRAS. The UniTESK tools are available under the free license. Experience of industrial application of UniTESK is fused into the tools. Some test suites developed with UniTESK are included in official test suites for certification of industrial software. For example, the OLVER test suite [7] is one of the biggest MBT test suites in the world and yields to the only test suite developed within the framework of the Microsoft Interoperability Initiative [8].

2 UniTESK Usage Review

Let's consider the most interesting examples of UniTESK application and experience gained in them.

The first application of UniTESK was in the project supported by Microsoft Research on development of the MBT test suite for IPv6 implementation [9]. The project started in 2000. At that time UniTESK was at the beginning of its development, so a simplified (light) implementation of API testing in C was used – CTESK-light. In spite of the tool instability, it allowed creation of the effective test suite that detected defects, which were not detected by other test suites. It was the first experience of using contract specifications for telecommunication protocol testing. It was demonstrated that contract specifications in combination with the technique of testing systems with asynchronous interfaces developed within UniTESK [10, 11] allow creating effective tests (they detected more defects, consumed less space and required less effort for development and maintenance than tests developed with traditional technologies). However, the experience of protocols testing showed that besides the post-conditions of the functions in the form of predicates it is useful to have executable models when testing protocols.

One of the first experiences of using UniTESK implementation for testing Java API [12] was the project on testing of Java run-time infrastructure developed as an alternative to the popular Java-platforms. The development of the models and the tests was not a problem since the interfaces were well documented. In addition to Java interfaces, the target system contained also the interfaces in C++, but they also were not a big problem since UniTESK architecture provides the layer of mediators-adapters. The problems revealed when the actual testing started. MBT test suite with online test generation is a fairly complicated program that has strong requirements to the execution platform. In this case, the SUT itself was the execution platform which still was not stable at that time. As a result, the test suite indicated the presence of defects "everywhere", which, in turn, was of no help to the developers.

The significant application of UniTESK on Java platform (JavaTESK) is the project on testing of infrastructure of the distributed information system of one of Russian major mobile telephony provider. This project is still in progress. The possibility of formal and rigorous specification of the components interfaces became the main advantage of UniTESK for the customer in comparison with the other tools for functional testing. Hundreds of components were formally specified and tested with UniTESK. By the end of the first year of using UniTESK the positive effect appeared in shorter time of integration of new versions of the distributed system. However, a serious problem revealed. In the previous UniTESK applications the requirements to most interfaces were defined by standards and other well-developed documents. But here the level of components documentation often appeared to be insufficient for creation of consistent specifications. Recovery of documentation or requirements to interfaces in the systems of such size becomes almost unsolvable task, which often makes it impossible to use MBT in corpore. Possible solution of this problem will be briefly discussed in Conclusion.

The largest example of UniTESK application is the OLVER (Open Linux VERification) project [7] fulfilled in 2005-2007 under support of the Russian Ministry of Education and Science. The goal of the project was to create formal specifications of interfaces defined in the Linux Standard Base (LSB) standard or in LSB Core – the central part of this standard, to be more exact. The LSB Core includes the most important libraries of OS Linux which implement most of the POSIX standard. The rigorous description of the LSB standard and the test suite capable of a high-quality checking of conformance of any Linux library implementation to the requirements of the standard is a very powerful tool for providing portability of OS Linux applications from one Linux distribution to another. The portability problem is very critical in the Linux ecosystem, since several hundreds of very different distributions are available. The project results are open [7]. The contract specifications of more than 1500 interfaces in C were created. Naturally, the CTESK tool was used for modeling and test generation. In this project, the problems in the standards were also revealed: in LSB (ISO/IEC 23360) and in The Single UNIX Specification containing the POSIX.1 standard (aka IEEE Std 1003.1, aka ISO/IEC 9945, aka The Open Group Base Specifications Issue 6) as its significant part. The developed test suite is included into the package of the certification tests of the international consortium The Linux Foundation [13].

The experience of interface formalization for a large industrial standard and test suite development for such standard gave many lessons to learn. One of such lessons is importance of informational and methodological organization of such project. The amount of documentation and sources, especially with respect to multiple versions and variants for different hardware platforms, is huge. Besides, the development of the standard and development of interface implementations involve thousands of people around the world. It means that the documentation maintenance and availability is one of the most important concerns of the projects of such scale. On the organizational and methodical side, we faced the fact that the training of new employees and the specification and tests quality control require a lot of effort, and the quick achievement of the required professional level is still impossible. In other words, the scalability of the MBT projects in the part of increasing the number of specification and verification experts is one of the most complicated problems preventing MBT from wide introduction.

One of the methodical problems is the choice of the abstraction level for the model. More abstract models or models separated into two-three layers of different abstraction levels simplify the reuse of the models and tests yielding, however, the bigger and more complex test system. In the long term, it's better to have multilayer models, while in the short term the models close to implementation in the detail level (of course, if the implementation already exists) are more appropriate. A professional and experienced verification expert can find the balance between the abstract description of the behavior, for example, of a file system and specifics and details of the interface of its particular implementation. UniTESK provides special support for the separation of abstraction levels. In particular, the specifics of interfaces can be encapsulated in the mediator-adapter layer. The choice of the balance is determined by the long-tem plans on using and improvement of the models and the test suite. So, the work of such kind requires a broad experience and long-term planning skills, which can hardly be expected from ordinary test engineers.

The results of the OLVER project were used later on in the development of the test suite for the Russian real-time operating system OS2000/3000 [14]. This system provides two groups of interfaces. The first group meets the requirements of the POSIX standard, the second one – the requirements of the ARINC-653 international standard for the embedded and other safety critical systems. The definition of the adapter layer separating model and implementation representations of the interfaces provided by the UniTESK architecture significantly simplified the OLVER reuse in this project.

Along with the start of the OLVER project, the work on the UniTESK application to testing of microprocessor designs [15] has been started. Hardware units being parts of Russian microprocessors with the MIPS architecture and microprocessors with VLIM/EPIC elements became the systems under test in this case. The size of typical units in such microprocessors is several millions of gates. The tools required no significant modifications for specification and test generation since CTESK was used as the basis. Technically, binding CTESK to corresponding API of microprocessor model simulator is not a problem, because most simulators that work with modeling languages for microprocessors logic (HLD – High Level Design languages), for example, VHDL or Verilog, provide suitable interface to C programs. Pre-conditions semantics in contract specifications had to be slightly modified. They now describe not just the domain of input data, but rather the operation execution readiness conditions in the given time frame. The same as in the case of protocols modeling, the use of explicit models of the target device behavior (functionality) along with the post-conditions in the form of predicates appeared to be necessary.

Similar to the projects on verification of software systems, one of the main problems preventing MBT from introduction into practice (as well as many other verification methods) is the lack of documentation and other descriptions of functional requirements to components. However, the situation in microprocessors development is slightly better than in the case of software development, because in this case it is customary to build system and architectural models of instruction set semantics along with the HLD models. Elements of these architectural models can be used to fill the gap in the knowledge on behavior of some microprocessor design units [16]. It appears also relatively simple to implement parallel test execution on clusters. Typical size of the finite-state machine generated during test execution for one microprocessor unit is millions of nodes and dozens of millions of transitions. The algorithm of FSM generation and exploration on clusters with up to 200 nodes appeared to require just 10-15% overhead, i.e. scalability coefficient is close to 1.

It is important to mention the verification tasks that, on the one hand, could not be reduced to modeling with contract specifications, and, on the other hand, pushed forward the development of new MBT methods. In the first place, the task of compiler testing should be mentioned, as well as the task of testing microprocessor as a whole, the so-called “core testing”. The both cases are the tasks of system testing, where test data and test stimuli are submitted to a big “black box” (in our case, these are test programs submitted to the compiler or loaded into memory of the microprocessor simulator), and it is interesting to test not just everything, but some specific behavior modes or specific group of units. In the case of compiler testing, the OTK tool has been developed that was used for testing of optimizing Intel compilers and Simulink [17, 18]. It allows targeting on specific kinds of optimizations. In the case of microprocessor design verification, the MicroTESK tool [19, 20, 21] was developed. The main goal of this tool is checking of various situations appearing in the most complicated subsystems of memory control: TLB, cache and Memory Management Unit (MMU) as a whole.

3 Conclusions and Further Work

Let’s start with positive conclusions.

3.1 Positive Conclusions on Modern State of Using MBT

- The world experience is confirmed [22], MBT can be effectively used in industrial projects, and in comparison with the traditional testing MBT gives a unique advantage – many defects can be found in requirements, which are often much more expensive than the defects in implementation
- The achievable level of test coverage is significantly higher than the traditional one (even in comparison with the “white box” testing). Thus, in the case of using OTK for testing GCC compiler, the achieved test coverage was 95%, and in the case of the Intel compiler this level was 75% that was significantly higher than the level achieved by traditional tests [18].
- Although the multi-level structure of specifications (several levels of abstraction) is seldom used in practice, the explicit separation of adapters layer simplifies tests porting and maintenance and, vise versa, the lack of the corresponding level of adaptation makes test suite development significantly more complicated, which was demonstrated in the Microsoft Interoperability Initiative program [23]
- Online generation of test sequences with the FSM exploration method can be efficiently parallelized and allows using computational resources of clusters with just 10-15% overhead, at least in the case of microprocessor models testing
- The demand of MBT in the safety critical area increases. This tendency can be found in standards defining requirements to development processes for safety critical systems, for example, in DO178C [24] and in Common Criteria [25].

3.2 Negative Aspects of the Modern State in the MBT Area

- The main obstacle preventing MBT from wide introduction into practice is the absence of specifications/models in casual software development. That is, the lack of specifications is often not only the consequence of insufficient attention to specification development or the consequence of short resources. The main reason is often the lack of qualified specialists who are experts in the knowledge domain and at the same time can create specification/model necessary for test generation.
- If MBT is used in projects that do not involve MDD (Model Driven Development) approach, then the model development delays the appearance of first tests – this does not allow obtain tests early in the development. If MBT is used within MDD, then the problems still remain, because different models required for development and for testing, in particular, for generation of different artifacts of the test suite. It is often considered as unacceptable additional cost, while with proper planning many components of the development models can be reused during test generation as demonstrated, for example, in M. M. Chupilko paper [16].
- Bilingual test generation systems like UniTESK and first versions of SpecExplorer [26], specification notations even close to conventional programming languages, for example, JML [27] make deployment of such systems difficult. Bilingual notations require special training of the staff and need permanent and expensive maintenance. Still note that the modern object-oriented languages already have advanced means for writing specifications just in the same language [26-31].

3.3 Directions of Further Works

- A variety of modeling paradigms should be used in various project contexts, in particular, contract specifications, various types of executable models, for example, finite-state machines, Kripke structures, etc. [32]. It is not obvious that the transformation of models from one paradigm into another one will bring real benefit. Each of the model kinds is suitable for analysis of specific aspects of the system behavior, so we should not expect that, for example, a functional model will facilitate estimation of the execution time and memory required. However, obtaining some skeleton or a prototype of the model of one kind on the basis of another kind is quite possible.
- The development of various tools for modeling and specification description for the MBT purposes is required. In spite of the progress in the area of technologies for development of Domain Specific Languages (DSL), practically, the systems based on universal languages benefit from the large number of programmers knowing such languages. The same can be also said about monolingual systems – they overtake multilingual ones.
- Modern achievements in the area of static and hybrid static-dynamic analysis allow integration of these techniques into the MBT systems, at that, the models/specifications as well as software implementations should be the subject of this analysis.

- To overcome the problems with the extreme lack of specifications in real practice, tools for work with requirements and models (see, for example, [33]), in particular, with system models [34, 35] should be developed and deployed. For multicomponent systems, MBT tools should be integrated with the tools for architecture and process mining.

References

1. Bourdonov, I. B., Kossatchev, A. S., Kuliamin, V. V., Petrenko, A. K.: UniTesK Test Suite Architecture. In: FME 2002. LNCS 2391, pp. 77–88. Springer-Verlag (2002)
2. Kuliamin, V. V., Petrenko, A. K., Kossatchev, A. S., Bourdonov, I. B.: The UniTesK Approach to Designing Test Suites. Programming and Computer Software, 29(6), 310–322 (2003)
3. Bourdonov, I. B, Kossatchev, A. S., Petrenko, A. K., Galter. D.: KVEST: Automated Generation of Test Suites from Formal Specifications. In: Proceedings of Formal Method Congress, Toulouse, France, 1999. LNCS 1708, pp. 608-621 (1999)
4. Peleska, J.: Industrial-Strength Model-Based Testing – State of the Art and Current Challenges. Invited Talk. In: Petrenko, A. K., Schlingloff, H. (eds.) Proceedings Eighth Workshop on Model-Based Testing (MBT 2013), Rome, Italy, 17th March 2013. Electronic Proceedings in Theoretical Computer Science, 111, pp. 3–28. DOI: 10.4204/EPTCS.111.1 (2013)
5. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag (2003)
6. UniTESK technology, http://unitesk.ispras.ru
7. OLVER project, http://linuxtesting.org
8. Microsoft Interoperability Initiative, http://www.microsoft.com/openspecifications
9. Pakulin, N. V., Khoroshilov, A. V.: Development of Formal Models and Conformance Testing for Systems with Asynchronous Interfaces and Telecommunications Protocols. Programming and Computer Software, 33 (6), 316-335 (2007)
10. Khoroshilov, A. V.: Specification and Testing of Components with Asynchronous Interfaces. Candidate's thesis, Moscow (2006)
11. Kuliamin, V. V., Petrenko, A. K., Pakulin, N. V.: Extended Design-by-Contract Approach to Specification and Conformance Testing of Distributed Software. In: Proceedings of WMSCI'2005, Orlando, USA, July 10-13, 2005. Model Based Development and Testing, v. VII, pp. 65-70 (2005)
12. Bourdonov, I. B., Demakov, A. V., Jarov, A. A., Kossatchev, A. S., Kuliamin, V. V., Petrenko, A. K., Zelenov, S. V.: Java Specification Extension for Automated Test Development. In: Proceedings of PSI'01. LNCS 2244, pp. 301-307. Springer-Verlag (2001)
13. The Linux Foundation Consortium. LSB Certification Test Suite, http://ispras.linuxbase.org/index.php/LSB_Certification_System
14. Maksimov, A. V.: Requirements-Based Conformance Testing of ARINC 653 Real-Time Operating Systems. In: Proceedings of the Data Systems in Aerospace (DASIA 2010) Conference, 2010. ESA SP-682, ISBN 978-92-9221-246-9 (2010)
15. Ivannikov, V. P., Kamkin, A. S., Kossatchev, A. S., Kuliamin, V. V., and Petrenko, A. K.: The Use of Contract Specifications for Representing Requirements and for Functional Testing of Hardware Models. Programming and Computer Software, 33(5), 272–282 (2007)

16. Chupilko, M. M.: Developing Test Systems of Multi-Modules Hardware Designs. ISSN 0361-7688, Programming and Computer Software, 38(1), 34–42, Pleiades Publishing, Ltd. (2012)
17. Zelenov, S. V., Zelenova, S. A.: Model-Based Testing of Optimizing Compilers. In: Proc. of the 19th IFIP TC6/WG6.1 International Conference on Testing of Software and Communicating Systems – 7th International Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2007). LNCS 4581, pp. 365–377. Springer-Verlag, Berlin Heidelberg (2007)
18. Zelenov, S. V., Silakov, D. V., Petrenko, A. K., Conrad, M., Fey I.: Automatic Test Generation for Model-Based code Generators. In: IEEE ISoLA 2006 Second Intern. Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Paphos, Cyprus, pp. 68-75 (2006)
19. Kamkin, A. S.: A method of Automation of Simulation Testing of Microprocessors with Conveyer Architecture Basing on Formal Specifications. Candidate's thesis, Moscow (2009)
20. Kornykhin, E. V.: A Method of Automation of Testing Program Generation for MMU Verification. Candidate's thesis, Moscow (2010)
21. Kamkin, A.S., Tatarnikov, A.: MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors. In: Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012), May 30-31, 2012, Perm, Russia (2012)
22. MBT Survey, http://www.robertvbinder.com/docs/arts/MBT-User-Survey.pdf
23. Grieskamp, W.: Microsoft's Protocol Documentation Program: A Success Story for Model-Based Testing. In: Testing – Practice and Research Techniques. LNCS 6303, p. 7 (2010)
24. Adams, C.: Safety-Critical Software for Mission-Critical Applications to Get Boost with Release of DO-178C. Military & Aerospace Electronics, 10 (2010)
25. Common Criteria, http://www.commoncriteriaportal.org
26. SpecExplorer, http://research.microsoft.com/en-us/projects/specexplorer
27. The Java Modelling Language (JML), http://www.eecs.ucf.edu/~leavens/JML/index.shtml
28. Pakulin, N. V.: Integrated Modular Avionics: New Challenges for MBT. In: ETSI TTCN-3 User Conference and Model Based Testing Workshop, Bangalore, India, 11-14 June 2012 (2012)
29. Code Contracts, http://research.microsoft.com/en-us/projects/contracts
30. C++TESK, http://forge.ispras.ru/projects/cpptesk-toolkit
31. Kuliamin, V. V.: Component Architecture of Model-Based Testing Environment. Programming and Computer Software, 36(5), 289–305 (2010)
32. Kuliamin, V. V.: Multi-paradigm Models as Source for Automated Test Construction. In: Proceedings of the 1-st Workshop on Model Based Testing (MBT'2004, in ETAPS'2004), Barcelona, Spain, March 27-38, 2004, Electronic Notes in Theoretical Computer Science, 111:137-160, Elseveir, (2005)
33. ReQuality tool, http://requality.org/en/doc.en.html
34. Khoroshilov, A. V., Albitskiy, D., Koverninskiy, I. V., Olshanskiy, M. Yu., Petrenko, A. K., Ugnenko, A. A.: AADL-Based Toolset for IMA System Design and Integration. SAE Int. J. Aerosp. 5(2) (2012)
35. Systems Modeling Language (SysML), http://www.sysml.org