Volume 6, Issue 5, May 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Analysis of Model-based Testing Methodology for Embedded Systems

**Pravin Karmore***
Research Scholar, RTM Nagpur University,
Nagpur, Maharashtra, India

**Pradeep Butey**
Associate Professor, Kamla Nehru Mahavidyalaya,
Nagpur, Maharashtra, India

***Abstract— Testing is a most crucial part of quality improvement of any system. In recent decade, Model-based approaches which are the Model-Based Testing (MBT) and Model-Driven Development (MDD) are being explored for testing automation in embedded systems. The model-based testing (MBT) methodology has been developed with Model-Driven Architecture (MDA). This paper analyzed the model-based testing on Architecture Analysis and Design Language (AADL) model with Markov chain, the integrated framework for model-driven development and system-level design validation with a combination of ESTEREL and SystemC as well as the applicability and adoption of the Unified Modeling Language (UML) and UML Testing Profile (UTP) for deploying MBT in Resource-Constrained (RC) Real-Time Embedded Systems (RTES).***

***Keywords— Embedded Systems, Model-based Testing, Model- driven Development, Model- driven Architecture, UML, UTP.***

## I. INTRODUCTION

The design of embedded system i.e. system-on-a-chip (SoC) has many challenges. System integration is an important challenge, because SoC integration involves combining various components such as CPUs, DSPs, ASICs, memories, buses, etc. The more complex in structure and larger in scale embedded system becomes, the more challenging to design and verify functional and non-functional necessities of embedded software. A big challenge for embedded software engineering is to address this growing variety and complexity of the software in the embedded system and ensure sufficient product quality. In order to achieve this, structured embedded software engineering and automation are inevitable. In this context, model-based methodologies (e.g. MDD and MBT), are being explored for automation in embedded software engineering projects. Adapting such automation methodologies are expected to improve the time to-market constraints and reduce the cost of embedded software development and testing. This paper finds out the workability of different model-based and model-driven approaches for embedded system engineering. A model-driven architecture (MDA) has been presented to meet demands of software design and implementation by Object Management Group (OMG). OMG also introduced Unified Modeling Language (UML) and UML Testing Profile (UTP) methodologies for MDD and MBT respectively [1].

Modeling has a major role during embedded software development. Application logical design is separated from software implementation with MDA. Unified Modeling Language (UML) as a key MDA method is widely used to model software architecture, component, object, and relationships among them. Some non-functional properties of embedded software are real-time, reliability and safety were described by UML. Architecture Analysis and Design Language (AADL) is produced by SAE, which can depict component model and evaluate interactive behaviors among components and some non-functional properties of software architecture [2]. The author Yun-wei Dong described the Markov chain to use MBT for testing AADL model, and an MBT framework in their paper to test AADL model [3]. Padma Iyenghar presents a detailed description of the UTP artifact generation procedure based on the chosen System Under Test (SUT) and the model design. A precise set of the elements from the UTP concept group is defined and implemented for the UTP artifact generation [4]. Recently in the industry and academia simulation-based validation is widely used to validate the implementation of the designed system. In this approach, the SUT is excited by various test scenarios and its output is validated against that of a golden reference model. The efficiency of this methodology is highly reliant on the quality of the test suite used for the validation. Therefore, test generation and test coverage are important aspects of functional validation. Deepak A. Mathaikutty presented a model-driven approach for development and validation of system-level designs [5]. The code coverage is a way begin but not end the complete verification, hence it is common to have code coverage complemented by functional coverage. The functional coverage deals with the functionality of the design and checks if all the important characteristics of the design's functionality have been tested. In other efforts such as a functional fault model is used to define functional coverage and the test generation is geared towards detecting these design faults. Assertions are used to monitor the performance of the system under test and to identify interesting legal events for functional coverage [7].

 

## II. THE RELATED WORK

### *A. The Model-Based Testing*

The model-based testing (MBT) approach has been developed with MDA. MBT is applied in two ways, first is that software under testing is changed from model and test case is generated according to software model; second is to detect errors by testing model under testing in software model. MBT analyzes expected behaviors of states which compose the prediction of MBT in decision-making path of model, and test-cases are selected from paths [6]. Test model receives test case as input from MBT and produce output. The difference between expected value in oracle and output of MBT finds errors. MBT can save developing and testing cost using predicate software quality property and evaluate software behaviors whether meet software specification in advance.

MBT can reduce the states of system under testing by means of abstraction or exclusion to void state space explosion [8, 9].MBT is a modeling-oriented methodology, and there are numerous models to illustrate software behaviors, such as Finite State Machine, State-charts, the Unified Modeling Language (UML), Architecture Analyze & Design Language and Markov Chains. Research on MBT for AADL model is just in the start step, and some MBT methods support AADL to test software non-functional attributes of embedded software. The MBT process itself is divided into the following steps in the literature [10]:

(a) Modeling the System Under Test (SUT) and/or its environment,
(b) Generating abstract tests from the model,
(c) Executing the tests on the SUT and

Based on these steps, various MBT methodologies have been reported for different application domains [11] [12]. A number of MBT approaches discussed in the literature concentrate on generating test cases using models extracted from software artifacts [13] [2], i.e. step (b), while a few approaches concentrate on steps (a) and (d) [11]. Similarly, there are several MBT tools [14] which support steps (a), (b) and (d). However, concrete research work or tool support (e.g. using UML/UTP) for deploying MBT (i.e. step (c)) is missing not only in the literature but also in commercially developed MBT tools. This is especially true for RC-RTES.

### *B. ESTEREL Language*

ESTEREL is an imperative language, which is used for modeling synchronous reactive systems [15], especially suited for control-dominated systems. It has a variety of constructs to state concurrency, communication and preemption; whereas data-handling follows the method of procedural languages such as C. Esterel has two different specification styles namely imperative and equational. Its semantics can be understood as a Finite State Mealy Machine (FSM), but it must make sure understanding, so that a program generates the same output sequence for the same input sequence. Internally, the ESTEREL compiler transforms the control part of a program into a set of Boolean equations with Boolean registers.

### *C. ESTEREL Studio*

Esterel Studio (ES) [16] is a development platform for designing reactive systems, which integrates a verification engine for design verification, a GUI for design capture and code generators to automatically generate target-specific executables. The GUI provides modeling through the graphical state machine specification called SSM or the ESTEREL textual specification [15]. The most important element of ES is the formal verifier (esVerify), which verifies the designer's objective. It performs both assertion-based verification as well as sequential equivalence checking. User-defined properties (stated as assertions) and automatically extracted properties (out-bound, overflow, etc) are formally verified by generating the proper counter-traces (.esi files) to point up violations. Finally, ES performs multi-targeted code generation, which range from targets such as RTL (in VHDL/Verilog) to ESL (in C).

### *D. UTP and MBT*

Fundamentally, UTP provides a basis for systematic testing and integration particularly in UML-based development environments. However, unlike the extensive research for using UML in the MDD phase, especially for RTES, applicability of UTP and its usage for RTES is a rising technology. A significant reasoning for this could be based on the role of UTP. For instance, the UTP is used only to specify explicitly a typical architecture that can be associated with MBT, while the MBT framework automates the testing process. This implies that the UTP as such does not specify how to carry out the testing process [17]. Hence, there are several MBT approaches for automating the testing process and only a very few of them concentrating on the applicability and specification of the test framework using the UTP. For instance, to our knowledge, an add-on [18] for MDD tool [19] is the only available MBT tool which supports development of UTP-based test infrastructure, especially for RTES.

## III. THE FRAMEWORK OF MODEL-BASED TESTING

In runtime, software requirement specifications are implemented by the software components, and expected behaviors of software could be specify as component mode. Surveys on MBT approaches conducted in [2] [13], identify the advantages and pitfalls in the existing MBT techniques. Based on the conclusions in [13], it is clear that the MBT approaches are typically not included with the software development process, for instance the MDD phase. It is also clear that there is a need for integration tools which combine both the MDD and MBT phases for software engineering projects. For example, though tools such as [19] (e.g. using UML for MDD) and [18] (e.g. using UTP for MBT) together could provide combined model-based methodology for RTES scenarios. However, currently they do not provide support for

 

deploying MBT in RC-RTES. Thus it is evident that not only UTP-based test framework development for MBT is missing but also, integrated model-based approach and test automation (combining UML and UTP) for deploying MBT in RCRTES is also missing in the existing approaches.

### *A. An MBT for AADL Model*

AADL outlines embedded software architecture with components models and its behavior model. Component models illustrate logical relationship among components composition, and dynamic attributes of software are illustrated with component mode which present one or more component behaviors. The construction of component mode is done by AADL error model. A component could be in one or more AADL modes. One AADL mode of component is combination of its ADDL error models, and the transformation of AADL mode can mapping into a Markov Chains (MC) model. Markov Chains is regarded as the model under testing.

The component error model states how a component mode transfers to others among software AADL architecture. An error model provides a set of error states for a component or connection, together with transitions and properties to specify how the error state of a connection changes due to error events and error propagations [20]. So we can construct estimated Markov Chain Model (EMC) of software as per requirement specifications and error model of software components, see Fig. 1. The chance of component mode transition in EMC is prior probability, and could get from software ex-version or specifications. And afterwards creation of the test input pool and testing oracle according to the probabilities of AADL component mode transition [3].

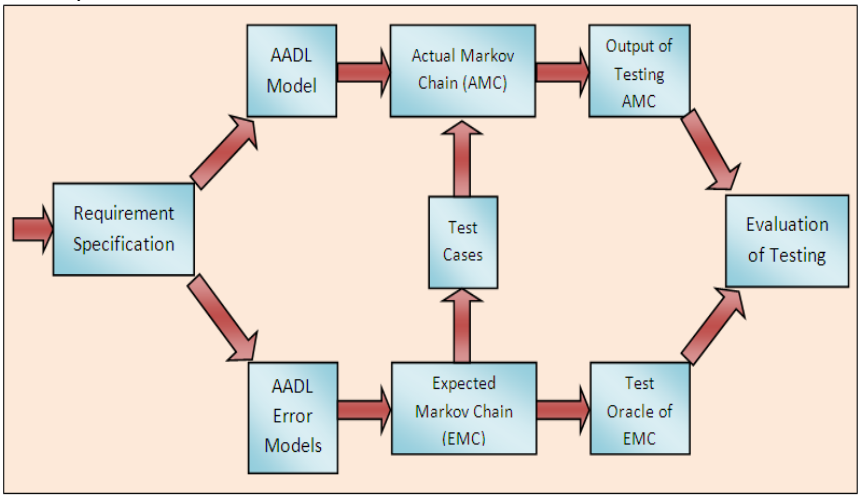


Fig. 1 Framework of MBT for AADL Model

### *B. Test Generation Methodology*

The validation flow shown in fig. 2, starts with a specification document that states the design requirements in a natural language such as English. A functional model (reference model) obtained from the specification in a language called ESTEREL [15]. This language easily provides abstracting away the implementation details and focuses on the functional behaviors. Its state machine formalism presents the functional model amenable to formal verification, which persuade for the intention of test generation through the development environment called Esterel Studio [16]. Properties are extracted from the English specification and modeled as verification monitors in ESTEREL.

These monitors are involved in (i) verifying the functional model and (ii) building conformance tests through Esterel Studio. These conformance tests are used to accomplish functional coverage of the implementation model. Furthermore, we apply the functional model with assertions to generate structural tests through Esterel Studio, which are used to achieve code coverage of the implementation. These structural and conformance tests are changed into concrete system-level tests that are used in the validation of the implementation model. The implementation model is also developed from the specification in the system level language SystemC [21].

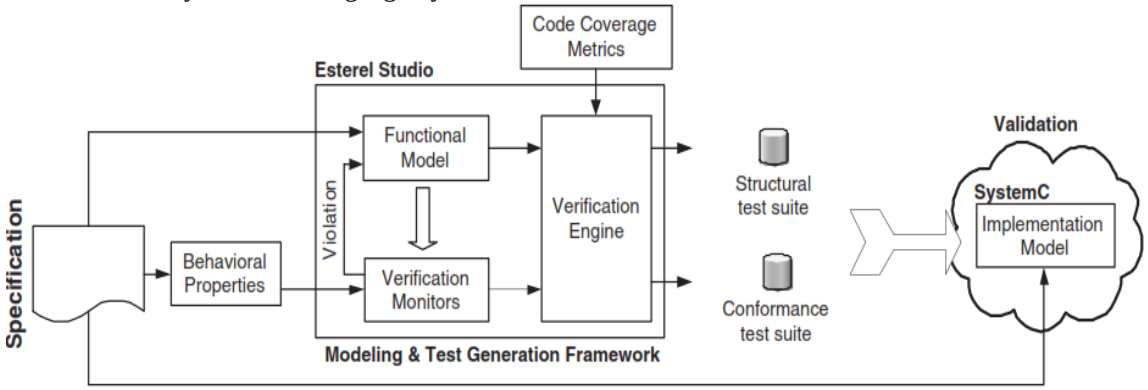


Fig. 2 Model-driven validation flow

 

Figure 3 shows 3-stage test generation methodology. In Stage 1, the user starts by developing a functional model in ESTEREL [15]. In Stage 2, the functional model and verification model are equipped using assertions for the purpose of test generation. The functional model is provided with assertions for code coverage, which outcomes in a structural coverage model. The verification model is provided with assertions for functional coverage, which outcomes in a conformance coverage model (same as functional model). In Stage 3, these annotate models are passed through the formal verifier (esVerify) in ES to build input scenarios in .esi files. The input scenarios are transformed into test benches in C along with the functional model using the code generation capability of ES. The test benches are compiled and executed on the C functional model and the outputs are recorded in .eso files. The input and output scenarios generated (.esi & .eso files) are used to construct the abstract test suite, which are transformed into system-level tests that attain structural or functional coverage on the implementation. Note that all models developed in our integrated framework are executables.

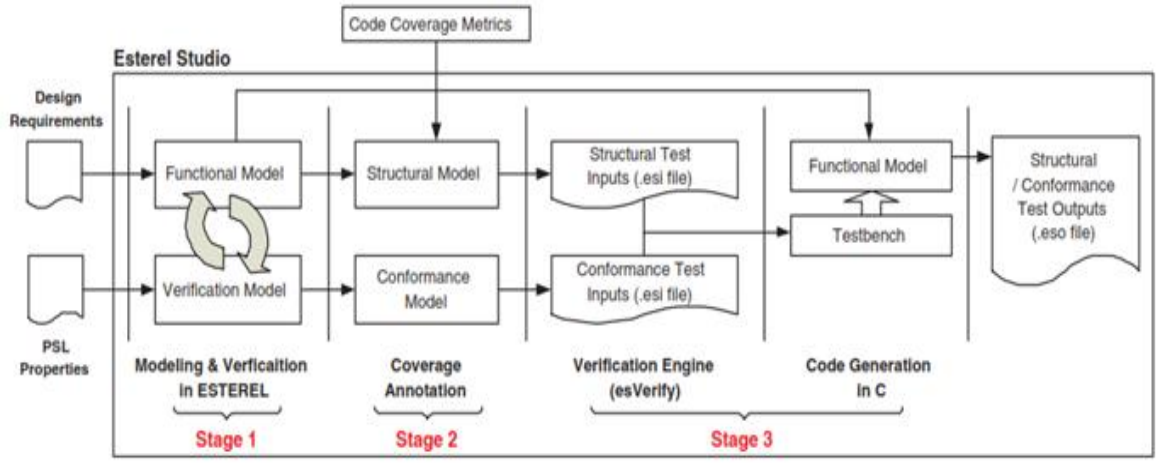


Fig.3 Framework of MBT generation methodology

An abstract test suite is a combination of test data and test cases having an abstract view on issues like communication, time and data. Tests generated from the functional and verification model are by nature abstract. These abstract tests must be changed into their concrete executable form in order to validate the implementation against the reference model. Figure 4 shows the executable test suite generated is employed in validating the implementation model developed in SystemC. The abstract test suite is created from the input and output scenarios obtained from ES, where these scenarios are extracted from the .esi and .eso files using a parser. The ES parser populates an Abstract Test Format (ATF) in order to generate the different test cases. The ATF is an XML schema for describing an abstract test case, which captures the number of execution steps needed and the input and output assignments for each step. The implementation model is also analyzed to extract the structural meta-information by a SystemC parser (SC Parser) that we implemented. The metadata describes the component characteristics that are utilized for exciting the component during testing, such as input-output ports for RTL components and read-write interfaces for transaction-level (TL) components. The SC parser is provided with the System Interface Format (SIF) schema as input along with the implementation model. The TestSpec Generator (TSG) transforms the abstract test suite into an executable test suite. Inputs to the TSG are the abstract test suite and .sid file generated from the implementation model. The output is an executable SystemC test suite.

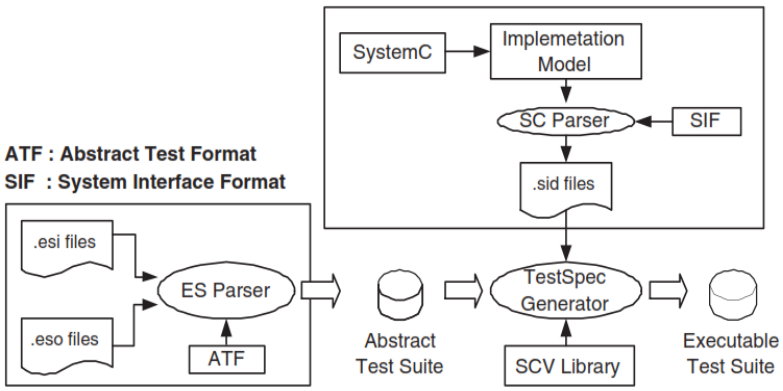


Fig. 4Executable test suite generation for SystemC validation

### *C. UML Testing Profile (UTP)*

The pragmatic development of test specifications and test models for black-box testing [22] [23] are provided by the UTP, where the internal events of the SUT are kept hidden. The UTP can be used stand alone for managing the test artifacts or in an integrated manner with UML for managing the system and test artifacts combined. The profile is based on UML 2.0 and extends its meta-model by the stereotype extensibility method.

 

The profile presents concept groups namely, test architecture, test behavior, test data and test time [1] [23]. The test architecture group describes a set of concepts to specify the structural aspects of a test context covering the test components, the SUT, their configuration, etc. A mapping of the UTP elements to the general schema of black-box testing [23] and UTP terminology is shown in Fig. 5. The test behavior group provides concepts to specify test behavior, their objectives and the evaluation of the SUT. The test data concept group is used to specify data used as stimuli (to the SUT). The time constraints and observations can be specified using concepts in the test time group in UTP. Together, these concepts describe a modeling language for visualizing, specifying, analyzing, constructing and documenting the artifacts of a test system [22] [1].

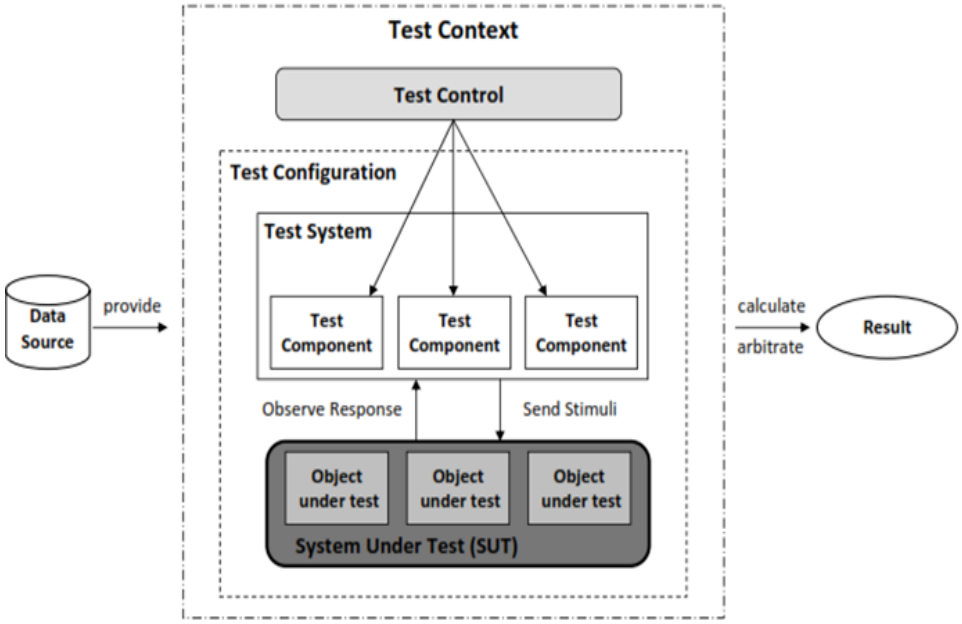


Fig. 5 General schema of black box testing and UTP terminology

### *D. Integrated Model-based Approach*

The combined model-based architecture and test framework for online MBT (introduced in [4]) in RCRTES is shown in Fig.6. As seen in Fig. 6, the two phases for RTES development and testing in approach are the MDD and MBT phases respectively. In the first step of the MDD phase, the requirements are specified as design model using UML. In order to derive a test framework automatically (MBT phase), a pre-defined system design model is necessary. The system design model provides the obligatory information for the derivation of the test framework skeleton. Based on the design model, the platform specific system code is obtained using an automatic code generation process. A MDD tool [18] is used for specifying the design model using UML as well as for the automatic code generation process in our prototype. During the compilation of the system code, an XML file is created with detailed data about the target and the developed embedded software [24]. The system code is executed on the RTES which runs a Real-Time Operating System (RTOS) bundled with a target monitor. The purpose of the target monitor is to send notifications about the target behavior to the host side using a debug communication interface [24]. A target debugger decodes the received trace data from the target monitor. The target debugger aids in communicating the test data between the test framework and the embedded target system.

During the MBT phase, given the design model and the selected SUT, a test framework is generated for deploying MBT in the RTES. The test framework comprises of three components, namely: (a) proxy test model (b) communication interface and (c) the UTP artifact. The proxy test model component comprises of a corresponding test module (e.g. class) in reference with each module (e.g. class) in the system design model. The proxy test model consists of only an abstract design model with no functionality inherited from the system design model. The proxy test model (on the host side at the design level) is used to mirror the test case execution process at the RTES. It also consists of the necessary infrastructure to convey the test data to the target via the target debugger and the target monitor as shown in fig. 6.

The test data is conveyed using a pre-defined format namely <(event) (source) (destination) (parameters)>. The test data is passed to the target debugger with the help of functions in the communication interface component of the test framework. The communication interface component helps in communicating (bi-directional) the test data, test results, etc between the test framework and the target debugger. This is implemented as a TCP/IP communication interface in prototype. The UTP artifact component is based on the concise set of UTP elements. It consists of the SUT under consideration, test driver to drive the SUT, test cases, test context and test configuration. In addition to these five elements based on UTP, two more components based on UML elements such as links and parts (instance of type class, i.e instances of associated classes of SUT) are created in the test context element in the UTP artifact. The links and parts, in addition to the UTP-based elements created in the UTP artifact, are used to aid the test case execution process. Once the test framework is generated, deploying MBT in RC-RTES using our approach and these UTP elements consists of a series of steps as illustrated in fig. 7.

 

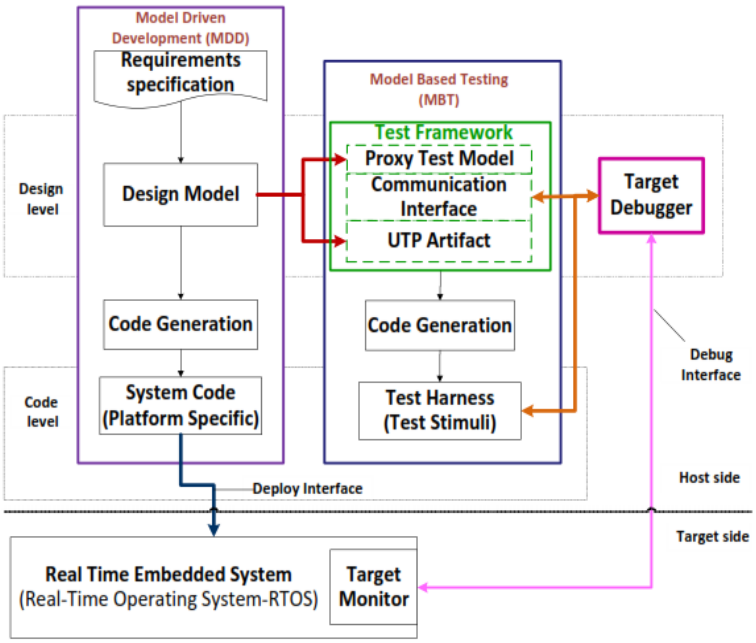


Fig. 6 Model-based test framework

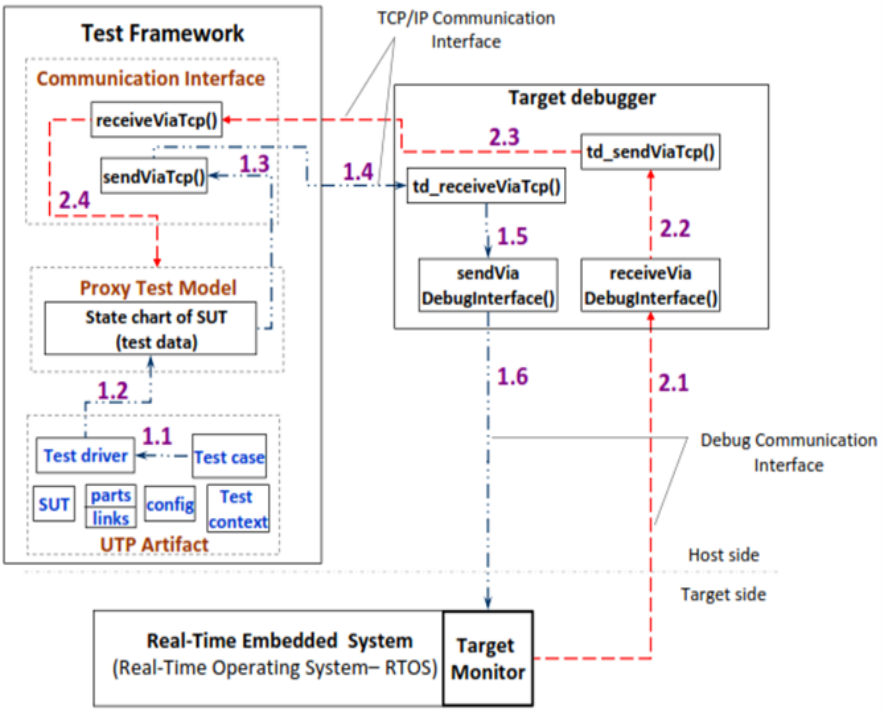


Fig.7 Deploying MBT

## IV. DISCUSSION

However, the UTP was introduced by the OMG several years ago; tool support for generating the test artifacts based on the UTP is currently very limited. Similarly, an integrated model-based approach combining both MDD and MBT, especially in the perception of RCRTES is missing. The testing of AADL model using Markov chain research for model-based testing has been in early stage, many such as test case generation and optimization, test coverage, test automation, analysis and assessment of result should be resolved [3]. An ESTEREL-based methodology for model-driven validation of SystemC designs through test generation is performed through coverage instrumentation and by using the counter-trace generation capability of Esterel Studio. The input and output test scenarios generated through ES are abstract tests used to achieve structural and functional coverage of the implementation. These are transformed into executable SystemC tests through a TestSpec generator and a system interface description of the implementation model [4]. The counter-trace generation limits the test generation to a single test solution even if there are multiple ways to test a particular implementation aspect. An integrated model-based approach and test framework for deploying MBT in RC-RTES using generation algorithm aiming towards model based test automation using UML and a concise set of UTP elements for RC-RTES [5].

 

## V. CONCLUSION

This paper analyses model-based testing methodology for AADL model. The analysis of the ESTEREL-based methodology for model-driven validation provides the major finding of abstract test suite to executable test suite. Model based test automation using UML and a concise set of UTP generates model-based approach and test framework for deploying model-based testing in real-time embedded systems. There is a requirement of empirical evaluation of the bug finding capability of different coverage metrics across test suites. Further, to support domains other than the real-time embedded systems using an extensible and generic framework can be developed.

## REFERENCES

[1] OMG, Model Driven Architecture (MDA), http://www. omg.org/mda/, 2002
[2] P. H. Feiler, D. P. Gluch, J. J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction, Carnegie Mellon University, 2006.
[3] Yun-wei DONG, Geng WANG, Hong-bing ZHAO, "A Model-based Testing for AADL Model of Embedded Software", 9th International Conference on Quality Software, IEEE, DOI 10.1109/QSIC.2009.
[4] Padma Iyenghar, Elke Pulvermueller and Clemens Westerkamp,. "Towards Model-Based Test Automation for Embedded Systems Using UML and UTP", IEEE, ITFA, 2011.
[5] Deepak A. Mathaikutty, Sumit Ahuja and Ajit Dingankar, "Model-driven Test Generation for System Level Validation". IEEE, 1-4244-1480, 2007.
[6] A. Pretschner, Model Based Testing, ICSE'05, May 15– 21, 2005.
[7] A. Ziv, "Cross-product functional coverage measurement with temporal properties-based assertions", proceedings of the conference on Design, Automation and Test in Europe, 2003.
[8] K. EI-Far, A. Whittaker, "Model-based Software Testing", Eneyclopedia on Software Engineering, Wiley, 2001.
[9] L. Alpelbaum, J. Doyle, "Model-based Software Testing", Software Quality Week Conference, May 1997.
[10] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann, 2007.
[11] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in Proceedings of the 21st international conference on Software engineering, ser. ICSE '99. New York, NY, USA: ACM, 1999, pp. 285–294.
[12] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in 2007 Future of Software Engineering, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 85–103.
[13] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A survey on model-based testing approaches: a systematic review," in Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies, ser. WEASELTech '07. New York, NY, USA: ACM, 2007, pp. 31–36.
[14] List of commercially available MBT Tools. http://www.cs.waikato.ac.nz/research/mbt/Tools.pdf, Jul. 2011.
[15] G. Berry, "The foundations of ESTEREL," Proof, Language and Interaction: Essays in Honour of Robin Milner, pp. 425–454, 2000.
[16] ESTEREL Technologies, "Esterel Studio," http://www.estereltechnologies.com/products/esterel-studio/.
[17] P. Krishnan and P. Pari-Salas, "Model-Based Testing and the UML Testing Profile," in Semantics and Algebraic Specification, ser. LNCS, J. Palsberg, Ed. Springer, 2009, vol. 5700, pp. 315–328.
[18] IBM Rational Test Conductor Add-on. http://www.btc-es.de/, Jul. 2011.
[19] IBM Rational Rhapsody Developer, version 7.5.1. http://www.ibm.com/software/awdtools/rhapsody/, Jul. 2011.
[20] SAE AS-2C Subcommittee, Architecture Analysis and Design Language (AADL) Annex Volume 1-E, http://www.sae.org, June 2006.
[21] OSCI Group, "SystemC Website," http://www.systemc.org/.
[22] P. Baker, Z. R. Dai, J. Grabowski, O. Haugen, I. Schieferdecker, and C. Williams, Model-Driven Testing: Using the UML Testing Pro?le. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.
[23] Z. R. Dai, "An Approach to Model-Driven Testing - Functional and Real-Time Testing with UML 2.0, U2TP and TTCN-3." Ph.D. dissertation, 2006.
[24] P. Iyenghar, C. Westerkamp, J. Wuebbelmann, and E. Pulver- mueller, "A model based approach for debugging embedded systems in real-time," in Proceedings of the tenth ACM international conference on Embedded software, ser. EMSOFT '10. NY, USA: ACM, 2010, pp. 69–78.

 