Journal of Theoretical and Applied Information Technology
10th October 2015. Vol.80. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A COMPREHENSIVE ARCHITECTURE FOR DYNAMIC EVOLUTION OF ONLINE TESTING OF AN EMBEDDED SYSTEM

SMT. J. SASI BHANU[1], A. VINAYA BABU[2], P. TRIMURTHY[3]

[1] Assistant professor, Dept. of Computer Science and Engineering, KL University, Guntur, India

[2] Professor, Dept. of Computer Science and Engineering, JNTU, Hyderabad, India

[3] Professor, Dept. of Computer Science and Engineering, ANU, Guntur, India

E-mail: [1]sasibhanu@kluniversity.in, avb1222@gmail.com, profpt@rediffmail.com

ABSTRACT

Embedded systems which are used for monitoring and controlling safety critical systems are to be evolved dynamically, meaning changes required either for command language interface, or to the Embedded systems software must be undertaken while the system is up and running. When the changes are made, the same are to be tested thoroughly before the changed code is made paramount. Different types of testing must be carried with the test cases initiated from the HOST. The testing must be carried while the embedded system is up and running. The kind of testing that must be undertaken sometimes cannot be foreseen. There is a necessity of dynamically evolving the very testing itself and then carry with testing with the help of test cases initiated from the HOST. In this paper a comprehensive dynamically evolvable online testing architecture has been presented and also a comparison of the same with other possible architectures has also been presented.

Keywords: *ES architectures, Dynamic Online Testing, monitoring and controlling safety critical systems, Software evolution*

1. INTRODUCTION

Embedded Applications are a different class of applications which throw several challenges especially related to Testing. The testing process to test the embedded applications involves testing individually Hardware, Software and both the Hardware and Software together. The process of testing an embedded application is rather complex and needs a detailed study. The process of testing an embedded system is not even streamlined as yet.

Development and testing of embedded software is especially difficult because it typically consists of a large number of concurrently executing and interacting tasks. Each task in embedded software is executed at different intervals under different conditions and with different timing requirements. Furthermore time available to develop and test embedded software is usually quite limited due to relatively short lifetime of the products.

Testing and debugging embedded systems is difficult and time consuming for simple reason that the embedded systems have neither storage nor adequate user interface. The users are extremely intolerable of buggy embedded systems. Embedded systems deal with external environment by way of sensing the physical parameters and also must provide outputs that control the external environment.

It is necessary to add software components related to dynamic evolution of syntax and semantics of the ES software of an embedded system that monitors and controls a safety or a mission critical system. The components added must be tested thoroughly before the same are made to be regular operational components.

No system is full proof. Failures and faults will occur. Upgrades to the existing software modules must be undertaken. New modules shall have to be added due to the need to add more functionality. It is also possible that some random errors may appear and it is necessary to know and fix those errors. Sometimes one can notice the degradation in response time and some of the devices might be malfunctioning. One can notice that the actuators malfunction and the control mechanisms that have been built into ES system have not been quite functioning. There could be many other such

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

reasons that makes it necessary for making changes to the embedded system and to those components that are added, deleted and modified which are related to dynamic syntax and semantic evolution of the embedded system leading to the necessity undertaking the testing.

The changes to the software have to be undertaken while ES system is up and running due safety and mission critical reasons. The embedded system must be tested thoroughly to ensure proper running of the same. The testing must also be undertaken when the system is up and running. The testing undertaken must be online, meaning testing undertaken while the application system is up running. The online testing have to be conducted without compromising on any of the issues.

Online testing involves addition of various test processes for undertaking different kinds of testing. Online testing requirement is an additional set of processes over and above the actual application related processes.

Online testing may be either related to hardware or software. A test environment must be set so that hardware testing can also be carried using the test cases initiated from the HOST. Several methods can be used for undertaking the testing at HOST or at the TARGET or both. Methods such as Scaffolding, Assert Macros, Instruction set simulators and third party tools are used for testing at the HOST, Logic analyzers are used to undertake testing Hardware, in-circuit emulators and monitors are used for testing ES software with the test cases initiated from the HOST. None of the methods are useful for carrying online testing using the processes that must be dynamically evolvable and using the processes that can test the changes carried to syntax and semantics while the software is up and running.

Online testing has to be carried due to the following reasons:

1. Decrease in response time
2. Decrease in the throughput
3. Non commencement of Actuating functions
4. Erratic sensing of the Inputs
5. Improper output displayed on to the output device
6. Memory conflicts
7. Conflicts in use of resources
8. Proper receipt of data transmitted by the HOST
9. Data out of range
10. Shared data problem
11. Failure of inter task communication
12. Existence of shared data problem
13. Adding new Tasks
14. Updating the existing tasks
15. Deleting existing tasks
16. Other types of testing requirements

2. PROBLEM DEFINITION

Testing of semantically evolvable embedded systems has to be carried on line. The testing system must itself be evolved dynamically. The testing of the embedded system meant for monitoring and controlling safety or mission critical system must be undertaken while ES software is up and running.

Testing must be carried whenever something faulty functioning within the embedded system is noticed to find the reasons for faults and take corrective actions. Examples of faults happening within the embedded systems include malfunctioning of sensing and actuating mechanism.

An online testing system must be evolved dynamically as it is not possible to pre-identify all kinds of testing that needs to be undertaken right at the time of designing the embedded system. Test processes must be added and test cases that are to be tested by the test processes must be identified and attached to the test processes dynamically. The mechanism required for undertaking testing must be effected dynamically.

Online testing must be undertaken along with the running system. The ES software cannot be shut down for want of making changes to the ES software. Both the ES System and the Testing system must be co-existing

Online testing must be carried whenever the ES software is evolved semantically or syntactically. Testing must be carried when new tasks are added, existing tasks are updated or deleted to ensure that the modifications to ES software is up and running.

Architectural models are required that include all the components related to online testing that must be co-existing along with the components that are related to syntax and semantic evolution and the components that are related to the ES software which is updated through changes initiated from the HOST.

3. LITERATURE SURVEY

The use of the embedded system can be presented in terms of usage models. Usage models can be developed using the user profiles, stimulus-response, User behavior profiles, UML models etc. Test cases are generated using the usage models which are kind of graphic structures. A model as

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

such characterize the uses of a software [G.H. Walton, et. al., 1995][1]. The data that should be used as input to be fed for undertaking the testing of the embedded system can be derived from the usage model [Whittaker et.al. 1994][2]. [Prowell S. J et.al. 1999][3] have presented that each path in usage model represents a test case that must be tested. The test case sufficiency can be judged with the help of arcs contained in a the usage models and the test cases that have been failed during the testing. The state and arc coverage within the USAGE model determined by a probability distribution model, provides a basis to verify the test sufficiency [Wolf M et.al. 2000][4]. Markov chain's if constructed using the usage models and probability distribution of the usage of paths contained in the usage model makes the generation of test case more interesting [Whittaker et.al. 1993][5], [J Poore et.al. 1998][6].

[Matthias Riebisch, et. al., 2003][7] explained the process of undertaking statistical use testing by way of exploiting the UML based use cases and sequence models as they provide data and behavior which are very much necessary for modeling the test case generation and also undertaking the test automation.

A tabular template based test case generation has been presented by [Cockburn A et.al. 1999][8], [Frohlich P et.al. 2000][9], [Sergiy A. Vilkomir, et. al., 2008][10] explained the process of generating automated test cases based on modeling combinatorial dependencies between input parameters and using Markov chain techniques. Input combinations play an important role rather than stimulus sequences in testing of software programs whose usage pattern consists of only three stages, namely, entering input parameters, calculation and generating results.

[D M Cohen et.al. 1997][11], [M Grindal et.al. 2005] [12] suggests different combinatorial approaches for situations where the number of possible parameter values is too large for testing all input combinations. Combinatorics together with Markov chains automates test cases selection, execution and evaluation and allows applying statistical analysis to the testing process. Markov chains are the usage models, each path in the usage graph relate to a particular usage of the application.

[W T Swain et.al. 2005][13] explains the application of Markov chain model for the selection of test cases from independent input parameters. A specific value represents each state of the model for each discrete input parameter. State transition occurs once the corresponding value is assigned to the parameter.

An proposed an approach close to the "Conflict-free sub-models" approach of [M Grindal et.al. 2006][14], differing in reducing the number of new states and generation of all test cases using only one model.

Test dependencies are used to develop a connected graph which can be used as a usage model. [Yan Jiong, et. al., 2003][15] advocated the usage of constrained UML artifacts like use case diagrams, sequence diagrams and the execution probability of each sequence diagram for deriving software usage models. Usage model generated by projecting the messages in sequence diagrams onto the objects under test and associating probability of occurrence for each message. Testing carried using usage models estimate software reliability.

[F Basanieri et.al. 2000][16] proposes to combine message sequences from sequence diagrams and generate test cases by using partitioning method. [A Abdurazik 2000][17] adapts the traditional data-flow coverage criteria as test adequacy criteria in the context of UML collaboration diagrams. [L Briand et.al. 2002][18] have presented the way system test requirements are derived from use cases and sequence diagrams by considering the sequential and dependability relationships between use cases and sequence diagrams. [Peter Frohlich et.al. 2000][19] Automatically generates system-level test suites with a given coverage level by transforming use cases into UML state charts and mapping its elements to the STRIPS planning language.

Testability constraints are imposed on UML artifacts and deriving a Markov chain usage model from them to support statistical testing. [Binder R 1999][20], [Bruegge B et. al. 2000][21] explained the sequential relationships of business requirements to the use cases use the relationships to generate test cases. The sequential relations between use cases have been represented through activity diagrams with vertices as use cases and edges as execution sequential relations between use cases. Use cases with no relations can execute in parallel. The pre and post conditions will determine the next executed use case. (Tomohiko Takagi et. al., 2004) [22] advocated the usage of operational data such as users' activities, log files of the program, etc., for generation of usage distribution which is useful in building the usage model at a low cost. This is achieved by applying source code generation methods based on a state machine diagram by establishing a one-to-one relationship between each transition on the state machine

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

diagram and basic blocks in the skeleton code generated.

Construction of probability distribution of a usage model is done by inserting probes into source codes while generating skeleton codes. These probes collect operational data in the state transition sequences or execution frequencies of each transition. This enables software to collect operational data for building a usage model. Source code generation using state transition table [B Beizer et.al. 1990][23], [J Ali et.al. 1998][24].

(Erik Simmons 2005)[25] defines the usage model as the one that describes the interactions between the user and the system at a level that identifies the systems' benefits to the user and presents a structure for usage model that contains three separate tiers: supporting data, overview, and usage details which provide a common taxonomy across various teams and business units. .

Product usage is described by use cases, scenarios, and concept-of-operations documents [Cockburn A et.al. 2001][26], [Fairley et. al., 1994][27] which all can be used for the generation of test cases. 29[Runeson P et. al., 1998][28] proposed use cases to be either translated or extended into concept-of-operations model. A common structure and taxonomy for describing product usage are necessary in order to unify requirements engineering, planning and design processes across business units and promote reuse.

[Regnell B et.al. 1995][29] Described a Synthesized Usage Model comprising of actors, usage views, use case specifications, abstract interface objects, user and system actions and a data dictionary. Construction of usage model starts from the beginning of the project. As the model develops supporting data is collected from which usage summary is created. Addition of usage details completes the model. In addition to the coverage of product usage the usage model also contains data about the environment in which the product is used and

The timing constraints in sequence diagrams in generation of usage model for real-time software statistical testing have also been used. Timing constrains are expressed by four classes of syntactic constructs [H Ben-Abdullah et.al. 1997][30] namely, Timers to express maximal delay between two events in one process, Delay Intervals to express time intervals between two consecutive events in a process, Drawing rules and timing markers used to express timing constraints.

OMG extended UML with a framework for representing time and time related mechanisms [OMG 2001] [31] to support real-time software development. The timed scenarios can be used to validate timing assignment and verify timing consistency [R Alur et. al., 1996][32], [X Li et. al., 1999][33].

Sequence diagrams are formalized based on partial ordering of events. A sequence diagram is defined by a finite set of instances, finite set of send and receives events, finite set of messages and a set of timing constraints. A Use case is defined by a set of sequence diagrams and a set of preconditions of the use case specified with Object Constraint Language (OCL) [Warmer J et. al., 1999][34]. The usage of high-order Markov chains for constructing accurate usage models. Statistical testing overcome the shortcomings of the systematic testing in expressing software reliability. The effectiveness of the statistical testing is determined by the accuracy of the usage model. By using high-order Markov chain in which the immediate past state is also considered along with the current state in determining the probability of event occurrence the accuracy of the usage model is increased.

High-order Markov chain generates test cases that the normal usage structure can't reveal. It increases the accuracy of evaluation of software reliability and also increases the effectiveness of statistical testing. The importance of using high-order Markov chain which considers the test case which are used in the previous instance has been highlighted.

[Thomas Bauer, et. al., 2007][35] advocated the usage of sequence-based requirements specification in combination with model-based statistical testing for very high degree of automation from requirements document to statistical test report.

Sequence -based specification is a set of techniques for stepwise construction of black box and state box specifications of software systems. It helps in analyzing the completeness and consistency of the requirements with a stepwise construction of a traceably correct black box specification.

UML based statistical testing requires, acquisition of execution probability of the messages in the sequence diagram associated with use cases and projecting them onto the objects of the system under test to generate usage models[Lyu M R 1996][36].

In statistical testing of software, all possible uses of the software, at some level of abstraction, are represented by a statistical model wherein each possible use of the software has an associated probability of occurrence [J.H. Poore et. al., 1998][37]. Test cases are drawn from the sample

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

population of possible uses according to the sample distribution and run against the software under test. Various statistics of interest, such as the estimated failure rate and mean time to failure of the software are computed. The testing performed is evaluated relative to the population of uses to determine whether or not to stop testing.

[Lin fan et al. 2008][38] have presented a method for testing a Embedded product using usage models and statistical methods. They have shown the testing process for Radio communication based application. The Application chosen by them is non-real time and as such no control logic exists. The usage states of the application have been manually recognized and have shown the way the usage and test chains can be generated by constructing Usage Model Transfer Matrix and Usage Model Excitation matrix.

Several authors have proposed different approaches to conducting testing of embedded systems. [Jacobson et. al., 1999][39] and others have suggested testing of modules of embedded systems by isolating the modules at run time and improving the integration of testing. This method has however failed to support the regression of events.

[Nancy Van Schooenderwoert et al., 2004][40] and others suggested an approach of carrying unit testing of the embedded systems using agile methods and using multiple strategies. Testing of embedded software is bound up with test of hardware, crossing professional and organizational boundaries.

[Tsai W.T et al., 2001][41] and others have suggested END-TO-END Integration testing of embedded system by specifying test scenarios as thin threads; each thread representing a single function. They have even developed a WEB based tool for carrying END-TO-END Integration Testing. [Lee N.H et al., 2003][42] suggested a different approach for conducting integration testing by considering interaction scenarios since the integration testing must consider sequence of external input events and internal interactions

Regression testing [Tsai W.T et al., 2001][43] has been a popular quality testing technique. Most regression testing's are based on code or software design, Tsai and others have suggested regression testing based on Test scenarios and the testing approach suggested is functional regression testing. Tsai and others have even suggested a WEB based tool to undertake the Regression testing. Others have suggested testing of embedded systems by simulating the Hardware on the host and combining the software with the simulators. This approach however will not be able to deal with all kinds of test scenarios related to Hardware. The complete behavior of Hardware specially unforeseen behavior cannot be simulated on a host machine. A testing approach has been proposed based on verification patterns, the key concept of this being recognizing the scenarios into patterns and applying the testing approach whenever similar patterns are recognized in any Embedded Application. But the key to this approach is the ability to identify all test scenarios that occur across all types of embedded applications.

Combinatorial testing is a black-box technique that could dramatically reduce the number of tests as it is a highly efficient technique to detect software faults. The method generally followed in combinatorial testing is to derive test cases from input domain of the system under test. But, when the input domain is larger and the output domain is much smaller, it is preferable to go for testing the output domain either exhaustively 47 [Zhao, R. et al., 2007][44], [D. Richard Kuhn et al., 2004][45] or as much as possible.

Exhaustive testing [Kuhn, D.R. et al., 2006][46], [Kuhn, R. et. al., 2008][47] is out of question when many input variables exist and they act in several combinations. Pseudo Exhaustive testing aims at considering only those input combinations that are most likely to act together.

Several authors have developed genetic algorithms for generating Test cases [Berndt, D. et al., 2003][48], [B. F. Jones et al., 1998][49], [Kamal Zuhairi Zamli et al., 2007][50] which uses input domain and also specially meant for generating test cases for loaded systems.

For some types of systems like safety critical embedded systems, developing test cases drawn from output domain will be more appropriate than from input domain as it ensures that all or as many possible output combinations are thoroughly tested [D. Bala Krishna Kamesh et. al., 2011-1][51]

The criticality regions of the embedded systems generally involve all parameters of input domain. The primary issue is to generate a set of test cases based on requirements specification that can detect as many faults as possible at a minimum cost and time in the criticality regions of embedded system. Another [D. Bala Krishna Kamesh, et. al., 2012][52] Adjacent Pair-wise Testing method that helps generating the test cases in the criticality regions of embedded systems has been proposed.

Combinatorial testing is a highly efficient technique that could dramatically detect software faults. It is useful to find whether the system under test is particularly sensitive to certain input values

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

or specific combinations of input values. Combinatorial testing can detect hard-to-find software faults more efficiently than manual test case selection Another method [D. Bala Krishna Kamesh, et. al., 2012][53]. Three main types of algorithms are available to construct combinatorial test suites viz. Algebraic [Colbourn, C. J. 2004][54], Greedy [Bryce, R. 2007][55], [Cohen, D.M., 1996][56], [Lei, Y. Kacker et al., 2008][57], and Heuristic search [Cohen, M. B et. al., 2008][58] algorithms. The method generally followed in combinatorial testing is to derive test cases from input domain of the system under test. Since there are often too many combinations of input parameters, it may be impossible to test all possible combinations for the system under test.

Some [D. Richard Kuhn et. al., 2011][59] combinations of input domain never occur in practice and some faults [R. Kuhn et al., 2008][60] are triggered only by unusual combinations – hence, the popularity of pair-wise testing, which is based on the observation that software faults often involve interaction between two parameters has been gained. Pair-wise combinatorial testing is an effective method which can decrease the number of test cases in a suite and is able to detect about 70% to more than 90% of faults.

Pair-wise combinatorial test suites could be generated by using different methods [Cohen D. M et al., 1997][61], [Lei, Y et al., 2002][62] and some of them lead to many invalid pair-wise combinations [Lixin Wang et al., 2010][63] of input parameters due to the input parameters relationships and degrade the performance of algorithm for generating test suite.

Combinatorial testing is a black-box technique that could dramatically reduce the number of tests as it is a highly efficient technique to detect software faults. The method generally followed in combinatorial testing is to derive test cases from input domain of the system under test. But, when the input domain is larger and the output domain is much smaller, it is preferable to go for testing the output domain either exhaustively [D. Richard Kuhn et al., 2004][64] or as much as possible.

For some types of systems like safety critical embedded systems, developing test cases drawn from output domain will be more appropriate than from input domain as it ensures that all or as many possible output combinations are thoroughly tested. Exhaustive testing of output domain is out of question when many input variables exist and they act in several combinations. Pseudo-exhaustive testing aims at considering only those combinations that will most likely result in failure conditions.

Several authors have developed genetic algorithms for generating test cases from input domain and those algorithms are specially meant for generating test cases for loaded systems. Consideration of output domain and the criticality regions of the embedded systems is more important when it comes to testing the embedded systems.

When it comes to the embedded systems exhaustive testing of the embedded systems in the critical regions is important. In the case of an embedded system that monitors and control within Nuclear reactor systems , commencement of pump operations when any of the sensed temperature is more than the reference temperature is critical. Criticality Regions of each of the Temperatures should be defined in the region of reference Temperatures.

There should be well proven architectural framework backing for undertaking testing of the changes that get dynamically evolved while the embedded system is up and running. Test processes that should be dynamically evolved must be able to undertake testing using the test cases from the remote HOST. [

[Sha et. al., 2001][65] has proposed an architecture that test an unknown process against a known process that executes correctly with reduced performance. The unknown process should be sufficiently tested by providing several test inputs and checking the timing and correctness of the results produced.

Sasi et. al., 2012-1][66] have presented a data repository model using which comprehensive testing of embedded systems can be carried. [Sasi et. al., 2012-2][67] have also presented a process oriented repository model which has been sued for undertaking the testing of stand-alone embedded systems.

4. INVESTIGATIONS AND FINDINGS

4.1 Over test Architecture

The overall architecture that support undertaking the testing of the embedded system is shown in the Figurer 1.

Journal of Theoretical and Applied Information Technology
10th October 2015. Vol.80. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

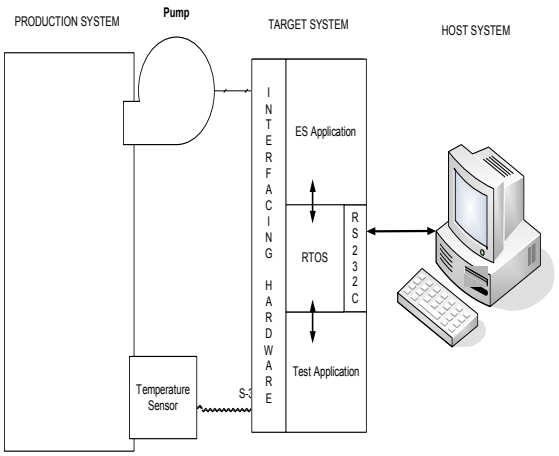


Figure 1 Overall architecture of testing the embedded system

It can be seen from Figure 1 that the Test application must be communicating with the ES application through a real time operating system and the commands for undertaking testing are initiated from a remote HOST.

4.2 Fully Blown testing Architecture

Figurer 2 shows Further exposition of the model shown in Figurer 1. From the model it can be seen that individual test processes are required for undertaking the online testing of the embedded system.

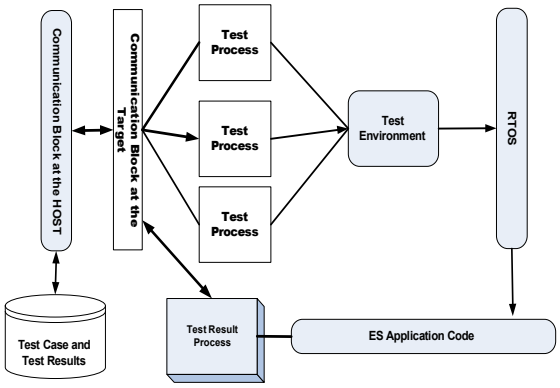


Figure. 2 Deeper exposition of Testing embedded system.

4.3 Expanded Testing architecture

Further exposition of testing embedded system considering the entire process of testing undertaken at the HOST and the TARGET is shown in the Figure 3.

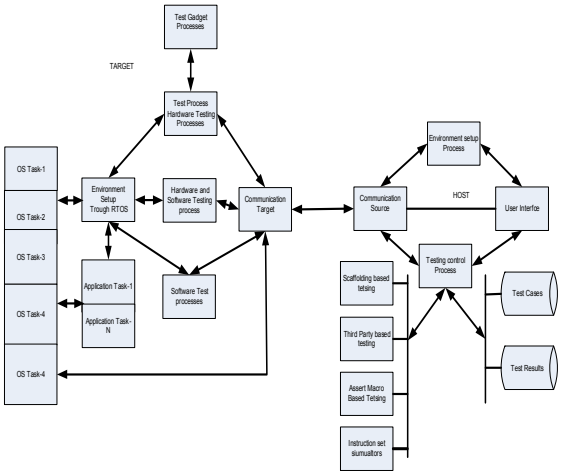


Figure 3 Total exposition of testing embedded systems

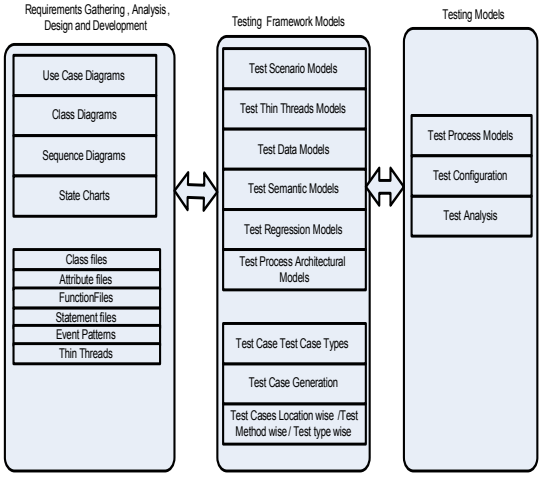
It can been seen from Figure 3 that Embedded systems are tested initially to the largest extent possible on the HOST by using the methods Scaffolding, Assert Macros, instruction set simulators and third party tools before the code is moved to target for testing. The command required for testing the target which includes either the hardware testing or software testing or both are transmitted from the HOST and the commands are used from undertaking the testing at the TARGET. The test cases and the test results are maintained at the HOST where the audit trails are conducted to find the sufficiency of testing that has been undertaken till the time audit trails are made.

4.4 Semantic Testing Architecture

A semantic model for undertaking the testing of the embedded systems and its related architectural model is shown in Figure 4. The architecture presented by them shows various software components that are required for testing the embedded systems. The model presented by then is quite suitable for testing of the stand-alone system. They have deliberated much on this architectural model. This model have has not included any of the component required for undertaking dynamic evolution of the embedded system.

Journal of Theoretical and Applied Information Technology
10th October 2015. Vol.80. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



Figurer 4 Semantic model for testing the embedded systems

4.5 Simplex Architecture

Simplex architecture includes a an extra component called decision logic which included into the overall semantic evolution. The testing of an update component before the old component is replaced by the new update component is undertaken as an overall software architecture. The architecture presented by them is shown in the Figure 5. The architecture is called as simplex architecture.

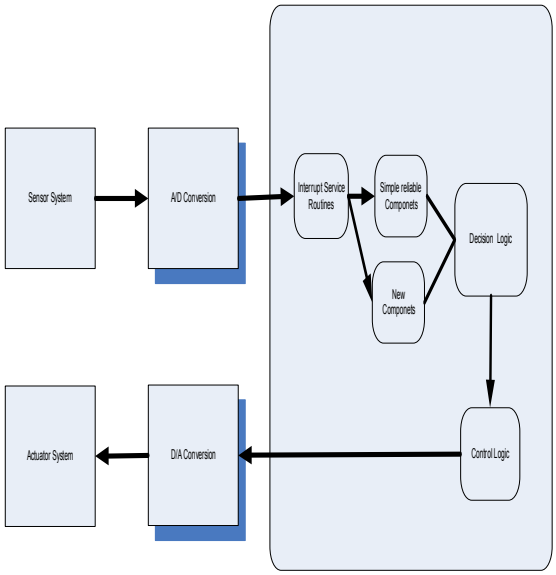


Figure 5 Simplex Architecture for online updates

In this architecture all critical tasks are performed by simple and verifiable components and the output of new components (Complex Components) which are counter parts to the simple verifiable components is fed as input to the simple components.

4.6 Process resurrection based testing architecture

Process resurrection based testing architecture proposed code safety without runtime checks for control system through usage of operating feature of Process protection. The replacing of the simplex code with Complex components and restarting of the application is achieved through the concept called Process Resurrection (PR). PR is a fast and efficient mechanism for restarting and replacing a process. PR mechanism is used to switch between the production mode and testing mode.

The new component image can replace the simple component and the mode can be changed from Testing to Production. In the production mode only the verified and tested components will be made to run. The runtime sub system must be able to reconfigure the code and switch between the production and testing in real time meaning the activities related to reconfiguration and switching must be predictable and schedulable. Figurer 6 illustrates the usage of the concept called process Resurrection. Process resurrection is a feature supported by most of the real time operating systems.

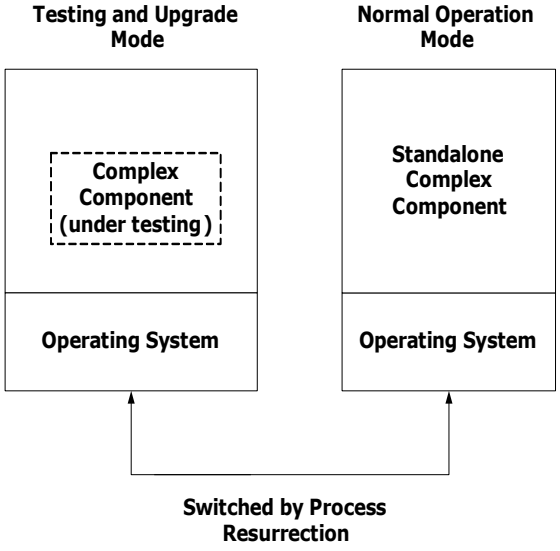


Figure 6 Process Resurrection For Mode Change Of The Embedded Systems

The mode switch is achieved through the process Resurrection function which will map the new code to the address space of the simple reliable component address space and also setting the control data a value realized by the control logic to indicate that a mode switch has been effected.

Journal of Theoretical and Applied Information Technology
10th October 2015. Vol.80. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The system runs in the normal operation mode when there is no need for testing or upgrade. In this mode only simple reliable functional modules shall be running. The switching overhead is very negligible and as such there is no CPU overhead and little extra storage is used to store the data related to changing the modes between productions and testing & upgrading. When online testing of new software is needed the PR feature converts the regular processes into Simplex enabled processes and the new software modules can be uploaded for testing.

When the new software is unproven and its features are needed then the system can be made to run at the cost of reduced response time of non-real time and non-critical tasks. When the new modules are believed to be running then the system is switched to normal mode of operation through the Process Resurrection. This architecture as such do not support dynamic evolution testing systems as the application system has to be temporarily to be put on hold while the testing is in progress which will not be allowed in the case of safety or mission critical systems.

4.7 Top View of dynamic evolution architecture

The top level view of an architecture of dynamically evolving embedded system is shown in the Figurer 7. The architecture has 3 Layers in It. In layer-1 a communication block exits which provides all the support required for communicating with the HOST.

All the commands are processed by the syntax evolution block situated in layer-2 and the commands are directed to the semantic, test, and syntax evolution systems and processes which are in layer-2. In layer-3 RTOS and various tasks that it runs to support any one of the evolution system.

The comprehensive architecture that incorporates the dynamic evolution of the online testing of the embedded system is shown in the Figure 8. The command line interface communicates with the HOST and receives all the commands from the HOST whether the commands relate to syntax evolution, semantic evolution or evolution of the testing. The correctness of the commands and directing the commands to the command processors where the semantics of the commands is verified and then the process that is related to the command is activated for execution.

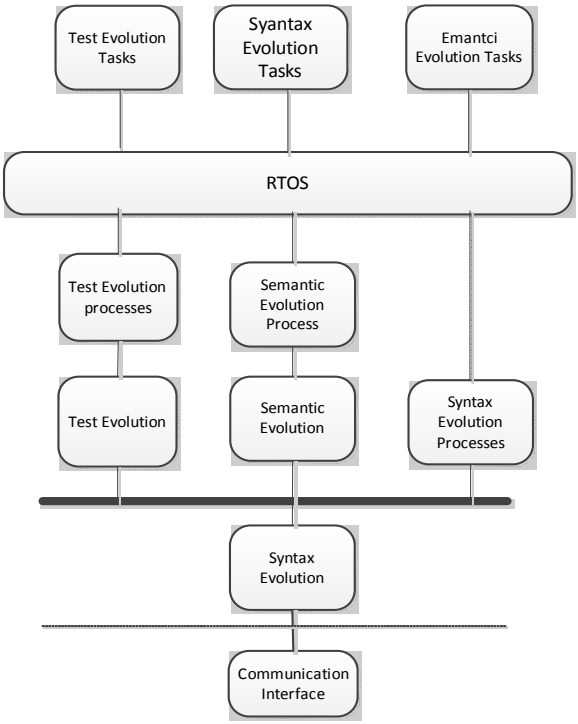


Figure 7 Overall dynamic evolution architecture

4.8 Comprehensive Architectural for dynamic evolution of online testing system

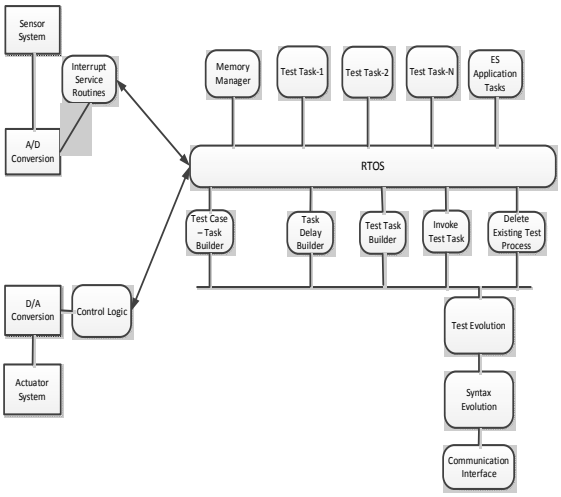


Figure 8 Comprehensive architecture for dynamic evolution of testing system

Testing can be made to be command driven. All the commands related to the testing can be grouped together into a command processor. The command processor directs the commands to process that undertaken the test concerned. In the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 8. Test Evolution block acts like a command processor. 5 different types of tasks are used for achving the dynamic evolution online testing process. The following are details of the tasks.

{1}. A task that builds and maintains the relationships between a command and a tasks that must be activated to undertake the related test

{2}. A builder that maintains the amount of delay that should be caused before the task is moved from blocked state to run state.

{3}. A test task builder creates a new test process task that actually undertake the concerned task.

{4}. A task that invokes the test process task for undertaking the actual test.

{5}. A task that deletes the existing task.

{6}. There can be many test execution process. The test execution process will return the results obtained after undertaking the test concerned. There can be any number of process tasks based on the type of test that must be conducted. For instance there can be individual tasks for undertaking testing for response time, throughput, proper reading of the input from the sensor, proper actuation of the devices, proper producing of the output into various devices etc. All the test execution tasks are in blocked state and the task is invoked when its related test case must be executed. This task is called as "invoke" Task

The invoking, sequencing and delaying any task is achieved through event processing support extended by RTOS All the tasks and processes will work under RTOS and sequencing and activation of various tasks is achieved through event handling support extended by the RTOS.

This architecture is truly dynamic that any number of test cases can be tested and only the test that is directed from the Host is undertaken. Additional test processes can be added, unwanted test processes can be deleted.

The testing to be undertaken is communicated by transmitting command related to the type of test that must be conducted along with the input data for undertaking the testing and the variable that should be used for returning the results.

5. COMPARATIVE ANALYSIS OF ARCHITECTURAL MODEL

A comparison is made to show the coverage of dynamic evolution for carrying online testing through different architectures. The comparison is shown in the Table 1 from which it could be seen that the comprehensive architectural model for online testing presented in this paper includes all the software components.

Table 1 Comparison of Testing related architectures

Parameter Serial	Parameter	Semantic Architecture	Simplex architecture	Process resurrection Architecture	Comprehensive Architecture
{1}.	Need for suspending the running system	√	X	√	X
{2}.	Update Testing	X	√	√	√
{3}.	New process testing	X	X	X	√
{4}.	Testing for deletion of a task	X	X	X	√
{5}.	Conducting different types of testing	X	X	X	√
{6}.	Comprehensive testing	X	X	X	√
{7}.	Cooperative Testing (Testing under co-existence of ES code and Test cooed)	X	X	X	√
{8}.	Creation of new test processes	X	X	X	√

6. CONCLUSIONS

In dynamically evolvable system changes are carried while the Embedded system is up and running. The changes are either carried to syntax or semantics of the embedded system. Additional components and methods are to be added for dynamically evolving the syntax and semantics of the embedded system. Every change intuited must be tested. There should be a dynamically evolvable testing system to undertaking the testing of the changes initiated to syntax and semantics of the embedded system. Test processes are to be added, deleted and modified online and the test processes are to be used for undertaking the testing of the changes intuited to the embedded system. A comprehensive architecture has to be designed that incorporates that considers evolving the testing processes online and use those processes for undertaking the testing using the test cases initiated from the HOST.

REFERENCES

[1] H. Walton,J.H. Poore and C.J. Trammell, "Statistical testing of software based on

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

usage model", Software Practice and Experience, Vol 25, No.1, PP. 97-108, 1995

[2] J. Whittaker and J. Poore, "Markov analysis of Software Specifications", ACM Transactions on Software Engineering and Methodology, Vol. 20, No. 10, , pp. 93-106, 1993

[3] Prowell, S.J., Trammell, C.J., Linger, R.C., Poore, J.H, "Cleanroom Software Engineering", Addison Wesley, 1st edition,1999

[4] Wolfgang Rosenstiel, Carsten Nitsch, R. Karlheinz Weiss, Thorsten Steckstor and , "Embedded System Architecture Design Based on Real-Time Emulation", Proceedings of the 11th IEEE International Workshop on Rapid System Prototyping, 2000

[5] Whittaker, J.A. and Thomason, M.G, "A Markov chain model for statistical software testing", IEEE Transactions on Software Engineering, Vol. 20, No. 10, pp. 812-824, 1994

[6] J.H Poore, J.P.Prowell., J.C. Trammell, R.C. Linger, , "Cleanroom Software Engineering: Technology and Process", Addison-Wesley, 1999

[7] Matthias Riebisch, Ilka Philippow and Marco Gotze, "UML-Based Statistical Test Case Generation", Proceedings of Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World by Springer-Verlag, 2003

[8] Cockburn, A, "Structuring Use cases with goals", Journal of Object-Oriented Programming, pp. 35-40, 1999

[9] Frohlich P. and Link J, "Automated test case generation from dynamic models", Proceedings of the 14th European Conference on Object-Oriented Programming, pp. 472-491, 2000

[10] Sergiy A. Vilkomir,Thomas Swain and Jesse H. Poore, , "Combinatorial test case selection with Markovian usage models", Fifth International Conference on Information Technology: New Generations, 2008

[11] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, , "The AETG system:An Approach to testing based on combinatorial design, IEEE Transactions on Software Engineering, PP. 437-444, 1997

[12] M. Grindal, J. Offutt and J. Mellin, "Handling constraints in the Input space when Using Combination Strategies for software testing", Technical report HS-IKI-TR-06-001, University of skovde, Sweden, 2006

[13] W.T. Swain and S.L. Scott, "Model-Based Statistical testing of a cluster Utility", Proceedings of the 5th International Conference on Computational Science, Atlanta,GA,USA, pp. 443-450, 2005

[14] M. Grindal, J. Offutt and S.F. Andler, "Combination testing strategies: A survey", Software Testing, Verification and Reliability, pp. 167-199, 2005

[15] Yan Jiong, Wang Ji and Chen Huowang, "Deriving Software Statistical Testing Model from UML Model", Proceedings of the Third International Conference On quality Software, 2003

[16] F. Basanieri and A. Bertolino, "A Practical Approach to UML-Based derivation of Integration Tests", Proc. 4th International Software Quality week Europe, Brussels, pp. 20-24, 2000

[17] Abdurazik and J. Offutt, "Using UML Collaboration Diagrams for Static Checking and Test Generation", Springer Lecture Notes in Computer Science, pp. 383-395, 2000

[18] L. Briand and Y.Labiche, "A UML-Based Approach to System Testing", Carleton University TR SCR-01-01-Version 2, 2002

[19] Peter Frohlich and Johannes Link, "Automated test case generation from dynamic models", Springer Lecture Notes in Computer Science, pp. 472-491,2000

[20] Binder, R, "Testing Object-Oriented Systems", Addison Wesley, 1999

[21] Bruegge, B. and A.H. Dutoit, "Object Oriented Software engineering: Conquering Complex and Changing Systems", Prentice Hall, 2000

[22] Tomohiko Takagi and Zengo Furukawa, "Constructing a Usage Model for Statistical Testing with Source Code Generation Methods", Proceedings of the 11th Asia-Pacific Software Engineering Conference, 2004

[23] B. Beizer, "Software Testing Techniques", Van Nostrand Reinhold, 1990

[24] J. Ali and J. Tanaka, "Generating Java Code from the Dynamic Model Based on Object Modeling Technique", Information Processing society of Japan, pp. 3084-3096, 1998

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[25] Erik Simmons, "The Usage Model: A structure for Richly Describing Product Usage during Design and Development", Proceedings of the 13th IEEE International Conference on Requirements Engineering, 2005
[26] Cockburn, A., "Writing Effective Use Cases", Addison-Wesley, 2001
[27] Fairley, R.E., Thayer, R.H., and Bjorke, P, "The concept of operations: The bridge from operational requirements to technical specifications", Proceedings of First International Conference on Requirements Engineering, pp. 40-47, 1994
[28] Runeson, P. and Regnell, B., "Derivation of an Integrated Operational Profile and Use Case Model", The Ninth International Symposium on Software Reliability Engineering, pp. 70-79, 1998
[29] Regnell, B., Kimbler, K. and Wesslen, A, "Improving the Use case Driven Approach to Requirements Engineering", IEEE Second International Symposium on Requirements Engineering, pp. 40-47,1995
[30] H. Ben-abdullah and S. Leue, "Timing Constraints in Message Sequence Chart Specifications", Proceedings of 10th International Conference on Formal Description Techniques, Japan, 1997
[31] OMG, "Response to the OMG RFP for schedulability, Performance, and Time", OMG Document Number: ad/2001-06-14, 2001
[32] R. Alur, G.J. Holzmann and D.Peled, "An analyzer for message sequence charts", springer Verlag, pp. 35-48, 1996
[33] X. Li and J. Lilius,"Timing Analysis of UML Sequence Diagrams", springer Verlag, pp. 661-674, 1999
[34] Warmer, J. and A. kleppe, "The Object Constraint Language: Precise Modeling with UML", Addison-Wesley,1999
[35] Thomas Bauer, Frank Bohr, Dennis Landmann, Taras Beletski, Robert Eschbach and Jesse Poore, "From Requirements to Statistical Testing of Embedded Systems", IEEE Fourth International Workshop on Software Engineering for Automotive Systems IEEE, 2007
[36] Lyu, M.R., "Handbook of software Reliability Engineering", McGraw-Hill Companies, 1996
[37] J. Poore, C. Trammell, "Engineering practices for statistical testing, crosstalk, The Journal of Defense Software engineering, pp. 24-28, 1998
[38] Lin Fan, Zeng Wenhua,Chen Guowu, "The Embedded Product Testing Using Cleanroom Statistical Method", 2009 World Congress on Computer Science and Information Engineering, 1999
[39] Jacobson, Booch G and Rumbaugh J, "The Unified Software Development Process", Addison Wesley, Reading, MA, 1999
[40] Nancy Van Schooenderwoert, "Taming the embedded Tiger-Agile Test Technique for embedded Software", IEEE Proceedings of the Agile Development Conference ADC, 2004
[41] W. T. Tsai, R. Mojdehbakhsh and F. Zhu, "Ensuring System and Software Reliability in Safety-Critical Systems", Proc. of IEEE ASSET, pp. 48-53, 1998
[42] Lee N.H and Cha S.D, "Generating Test Sequences from a set of MSCs", Computer Networks, 2003
[43] Tsai W.T, Bai X, Paul R and Yu L, "Scenario-Based Function Regression Testing", Proc. of IEEE COMPSAC, pp. 496-201, 2001
[44] Zhao, R. and Shanshan Lv, "Neural-Network Based Test Cases Generation Using Genetic Algorithm", 13th IEEE International Symposium on Pacific Rim, Dependable Computing, pp. 97-100, 2007
[45] D. Richard Kuhn, Raghu N. Kacker and Yu Lei, "Practical combinatorial testing", NIST Special Publication, 2010
[46] Kuhn, D.R. and Okun, V, "Pseudo-Exhaustive Testing for Software", 30th Annual IEEE/NASA Software Engineering Workshop SEW-30(SEW'06), pp. 153-158, 2006
[47] Kuhn, R. ,Yu Lei and Kacker, R, "Practical Combinatorial Testing: beyond Pair wise", IEEE Computer Society - IT Professional, Vol. 10, Iss. 3, pp. 19-23, 2008
[48] Berndt, D., Fisher, J., Johnson, L., Pinglikar, J. and Watkins, A, "Breeding Software Test Cases with Genetic Algorithms", IEEE Proceedings of the 36th Hawaii International Conference on System Sciences, 2003
[49] B. F. Jones, D. E. Eyres and H. -h. Sthamer, "A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing", The Computer Journal, Vol. 41, No. 2, 1998
[50] Kamal Zuhairi Zamli, Nor Ashidi Mat Isa, Mohamed Fadel Jamil Klaib and Siti Norbaya Azizan, "A Tool for Automated

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Test Data Generation (and Execution) Based on Combinatorial Approach", International Journal of Software Engineering and Its Applications, Vol. 1, No. 1, pp. 19-36, 2007

[51] D. Bala Krishna Kamesh, Vudatha, C.P. , Jammalamadaka, S.K.R., Nalliboena, and Reddy,L.S.S, "Automated generation of test cases from out putdomain of an embedded system using Genetic algorithms" 3rd International Conference on Electronics Computer Technology (ICECT), Vol. 5, pp. 216-220, 2011

[52] Kamesh DBK, "Comprehensive testing of embedded systems using refined cleanroom software methodology", Thesis submitted to Shri Venkateswara University, Gajroula, UP, 2012

[53] D. Bala Krishna Kamesh, Chandra Prakash Vudatha, Dr. Sastry KR Jammalamadaka, Haritha SV Grandhi, Vandana Lakshmi Nunna, Dr. Reddy L.S.S, "Automated generation of Test cases for testing critical regions of Embedded systems through Adjacent Pair-wise Testing", International Journal of Mathematics and Computational Methods in Science & Technology, Vol.2, No.2, pp. 10-15, 2012

[54] Colbourn, C. J, "Combinatorial aspects of covering arrays", Le Matematiche (Catania, pp. 121-167, 2004

[55] Bryce, R, "The Density Algorithm for Pairwise Interaction Testing", Journal of Software: Testing, Verification and Reliability, pp. 159-182, 2007

[56] Cohen D. M., Dalal S. R., Fredman M. L. and Patton G. C,"The AETG system: an approach to testing based on combinatorial design", IEEE Transactions on Software Engineering, pp. 437-444, 1997

[57] Lei, Y and Tai, K. C, "A Test Generating Strategy for Pair-wise Testing", IEEE Transactions on Software Engineering", pp. 1-3, 2002

[58] Cohen, M. B., Colbourn, C. J., and Ling, A. C. H, Constructing Strength 3 Covering Arrays with Augmented Annealing, Discrete Mathematics, pp. 2709-2722, 2008

[59] D. Richard Kuhn, Raghu Kacker and Yu Lei, "Combinatorial and Random Testing Effectiveness for a Grid Computer Simulator", NIST – National Institute of Standards and Technology, 2007

[60] Kuhn, R. ,Yu Lei and Kacker, R, "Practical Combinatorial Testing: beyond Pair wise", IEEE Computer Society - IT Professional, Vol. 10, No 3, pp. 19-23, 2008

[61] Cohen, D.M., Dalal, S. R., Fredman, M. L., and Patton, G. C, "Method and system for automatically", generating efficient test cases for systems having interacting elements", United States Patent, pp. 542-543, 1996

[62] Lei, Y. Kacker, R. Kuhn, D. Okun and V. Lawrence J, IPOG/IPOD: Efficient Test Generation for Multi-Way Combinatorial Testing, Journal of Software: Testing, Verification and Reliability, pp. 125-148, 2008

[63] Lixin Wang and Renxia Wan, "A New Method of Reducing Pair-wise Combinatorial Test Suite, Computer and Information Science, ccsenet.org, 2010

[64] D. Richard Kuhn, Wallace, D.R. and allo, A.M., Jr, Software Fault Interactions and Implications for Software Testing", IEEE transactions on software engineering, Vol. 30, No. 6, pp. 418-421, 2004

[65] Sha L, Lee K and, "Process Resurrection: a fast recovery mechanism for real-time embedded systems, Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium, 2005

[66] Sasi Bhanu. Vinaya babu A, Sastry JKR, "Scenario Based Comprehensive Testing of Embedded Systems using Data Models", International Journal of Advances in Science and Technology, Vol. 4, Iss. 6, pp. 34-38, 2012-1

[67] Sasi Bhanu. Vinaya babu A, Sastry JKR, "Scenario based Comprehensive Testing of Embedded Systems using Process Models", International Journal of Computer Information Systems, Vol. 4, Iss. 6, PP. 8-11, 2012-2.