International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Test Case Generation for Real-Time System Software Using Specification Diagram

Mani Padmanabhan[1]* **Prasanna Mani[2]**

[1,2] *School of Information Technology and Engineering,*
Vellore Institute of Technology, Vellore, Tamil Nadu, India
* Corresponding author's Email: mani.p@vit.ac.in

Abstract: Software testing of the real-time system (RTS) software based on specification diagram has a necessary sequence of parallel events for generation of test cases. In the model-based test case generation for RTS both automated and manual is limited in techniques as some situations inadvertently forget the simulation events. Many meta-heuristic techniques have solved the problem in RTS test cases generation. However, the technique seems to have much more focus on the optimum solution. This paper presents a methodology based on timed input-output event of specification diagram that supports a wide range of possible test cases for RTS application software. The proposed method applies for automated test case generation tool aims at validating the executable code and covers the systems behaviours in an optimized manner. The methodology is tested with three different RTS software such as client-server based real-time transaction, embedded based real-time sequence and sensor based real –time events. This proposed approach, based on the executable test cases, dynamically increases the productivity of the real-time systems.

Keywords: Software engineering, Software testing, Real-time systems, Software quality, Test case generation.

1. Introduction

Effective testing of the real-time system (RTS) is necessary to produce reliable systems. RTS controls an environment by receiving data, processing them and returning the result within specified time. RTS software behaviour is based on timing constraints [1]. RTS specification design has the combination of event components and process description in a diagrammatic way to understand the stimulus process. Testing the application software of RTS is a complex process because RTS programs are quite near to hardware device. The event sequence depends on the logical gates and processors. If the testing is poorly implemented, this results in a low-quality software and money wastage. Most often, a tester is given a set of tasks with the job of verifying the event that can be performed using the software [2]. Due to the application of conventional method of using a process based on timing priority, the testing of RTS is adversely affected. The multithreaded sequence between the systems component remains unidentified in the process of testing, which provides a safe hiding place to the bugs inside the application. Thereby, such bugs manage to trigger the specific path in the application. Test cases are described as triplets (Fi, D, Fo): the actual process of input (Fi), the changes made based on the input (D) and the actual output (F0) to be produced. Automated test case generation can reduce the bugs that arise in the combination of concurrency and RTS failures. However, the presence of many bugs during the real- time process also leads to customer dissatisfaction [3]. The Information Technology companies have acknowledged that testing methodologies have to be improved as much as possible so as to bring accuracy in RTS. The formal wide ranges of test case generation research fields have addressed various aspects of the process. Many

of these approaches focus on functional correctives, rather than the non-functional coverage.

Software testing is to be performed at different levels during the software development life cycle. During the verification tests, many runtime problems may be found, such as unexpected exceptions and incorrect or abnormal program behaviours. Since the testing team and the development team are independent of each other, problems found during the verification tests are generally passed on to the development team through textual descriptions as the problem description and log information, a tedious and inefficient process, cannot be directly used by the testing team. It is very difficult to reproduce the runtime problem. In order to strengthen the organization, RTS test cases are generated based on the software design, whereby, RTS failures can be reduced [7]. There is no systematic activity when the automation should start and end. The RTS automated testing activities can do their development simultaneously with RTS software development. The early development of test cases may reduce more bugs in product [7]. The delivery of the automated test suite should be done before the test execution phase so that the deliverables from the automation effort can be utilized for the current release of the RTS application.

RTS are generally used in the safety-critical area. This simulation transaction implies that there is a need for sequence verification methods for the removal of the bugs, failing which could lead to system failure. Software testing is a process to verify the software effectively by executing the possible test cases. The behaviour of RTS is time-sensitive; Therefore, RTS must verify the events before deployment [8].

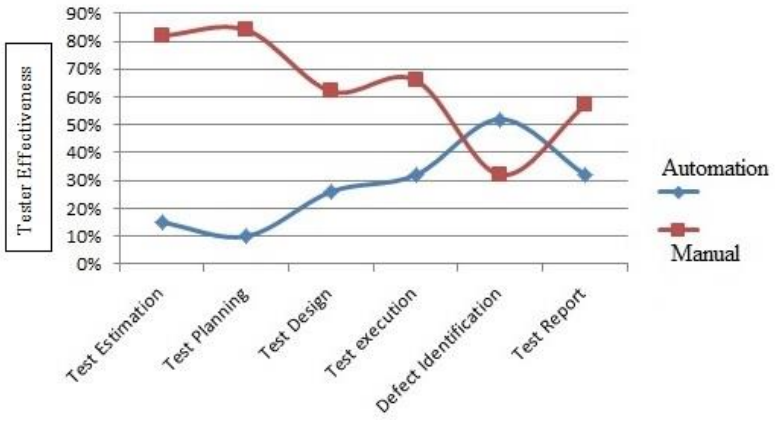


Figure.1 Automation in STLC

In the survey, software development respondents were asked to describe their current status of test automation. The majority of the practitioners did not automate their testing process. The similar result was reported by J. Jee et al. [8]. The percentage of automation process in the software testing lifecycle is shown in Figure 1.

This paper attempts to describe the flexible framework for producing possible test cases automatically for RTS. Test case generation from design specification of RTS has the added advantage of allowing test cases to be available early in the development cycle [3]. Software specification diagrams are suitable source of information for test case generation. In this research work, software design based test case generation process is proposed to reduce the testing effort and to improve the quality of the software. The main goal of this framework is to develop a possible test case automatically to reduce the bugs and avoid redundant test cases during RTS application testing. This proposed approach will be explained with three real-time applications and the results will be compared with the existing approaches. Background section discusses the related methodology for the test case generation. The proposed workflow for the framework and algorithm to convert the input specification to event flow graph [EFG] during the automated test case generation are described in proposed framework section. The experimental results for three different RTS such as automated water pumping system with activity diagram, Robot elevators lift with client-server technique, Tours reserving system based on embedded system software are compared in experimental section. The final section concludes with a discussion of future work.

2. Related Work

The development of software products follows a process where the development team is responsible for the development and test of the product and the testing group looks into the verification of products. This section presents the activities associated with test case generation for RTS. The methodologies to be followed in the approach are discussed in relation to existing approach. A wide range of studies on automated test case generation classified the RTS test case generation with real-time systems. Andrade WL et al., [1] presented an approach generating test cases for RTS based on a symbolic model. In this approach, test case generation strategy is an ability to handle data and time when compared to other approaches. The symbolic model addresses the issues such as nondeterminism and parallel composition although it has not directly addressed. Table 1 describes the related test case generation methodology based on tree and graph model.

E. Jee et al., [8] present their approach for Functional Block Diagram (FBD) to test suite. They have proposed a test tool for generating automated test stubs based on testable sequence diagram from behaviour specification. M. Chen et al., [6] have described novel test condition based test case generation for collaboration diagram. They have used a traditional data flow criteria and also have proposed a criterion for dynamic testing for message sequence path. Fujiwara et al., [9] have given the formal description of web application behaviours and data constraints with Eclipse Modelling Framework (EMF) class diagrams and Object Constraint Language (OCL) notation. Enoiu EP et al., [10] has used the UPPALL timed automata and timing clock for generation of test cases, where the symbolic model is used to represent a parallel sequence but symbolic data is not handled. The entire action in an event-recording automaton is associated with a clock. In order to take the timing aspect of the real-time system, the author extends the necessary test theory. The generated test cases are not based on the region graph.

The above-mentioned research works used automatic and manual methods to generate the test cases. And it is evident that the number of simulation path and components increases in the whole process, thereby leaving a space for more failures in RTS testing. Detection of all event flow in RTS is a major problem. The proposed research work differs from the below-mentioned research on the following grounds: the proposed framework will observe the event flow components from specification diagrams; then, the event flow will be converted to the Event flow Graph (EFG) with timing priority; and finally, the event flows are transferred to the stack stimulus to generate the possible test cases for the RTS specification. However, the approach tackles the challenges of test case generation of RTS with undetermined input specifications. The detailed framework components and algorithms are to be explained in the following subsection.

Table 1. Test case generation from specification diagram

Author(s) /Year	Input Specification Model	Root Model	Intermediate model	Coverage criteria
Wu Y-C et al., (2014), [3]	Functional block Diagram	Tree model	Invocation sequence tree (IST)	Full predicate, round trip path
S. Kansomkeat et al., (2010), [14]	Activity Diagram	Tree model	Condition classification trees	Transition, activity
Abou Trab MS et al.,(2013) [2]	Activity Diagram	Graph model	Activity graph	Activity path
A. Nayak et al., (2010), [16]	Sequence Diagram	Graph model	Structured control graph	self path
S. K. Swain et al., (2010),[5]	Sequence and state diagram	Hybrid	State activity diagram	sequence node, edge, path
Wang H et al., (2016), [17]	State Diagram	Heuristic	Scenario tree and colored petri-net	Sequential and concurrency

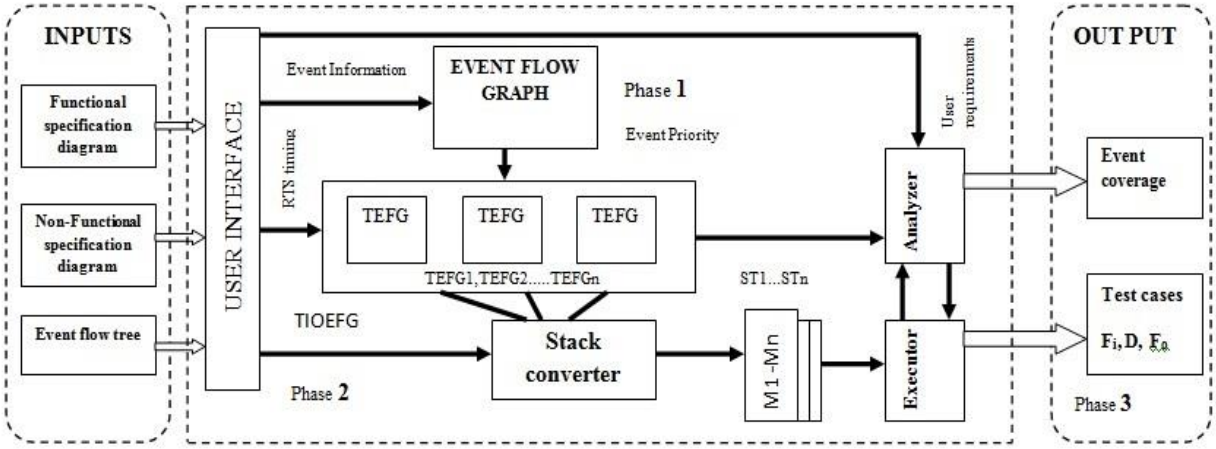
3. Proposed methodology

Real-time systems are related with our environment using time constrained events. In case the accuracy of the RTS is not guaranteed, it can lead to devastating consequences due to the functional misbehaviour or nonconformity in the specific time constraints. A timed event model for conformance test case of RTS is to be addressed in terms of both data and time symbolically using timed input-output event transaction system [TIOETS]. The TIOETS identifies the data event for symbolic analysis of timed aspects. Based on the

TIOETS, a framework named timed input-output event test case generator [TIOETG] is presented in this paper. The framework contains three primary components.

3.1 Event flow graph

A tool for generating a structural model of the specification into a graph is carried out through an Event flow graph (EFG). The primary purpose of the Event flows graph converter is to determine structural information from the given inputs notwithstanding graph representing relationships between events in the specification diagram using an algorithm and human inputs. This input data is converting into an EFG. The detailed workflow of TIOETG is shown in Figure 2.



Phase1: Event flow graph converter**; Phase 2:** Timing event flow graph converter**; Phase 3:** Test case generator.

Figure.2 Test case generator framework (TIOETG)

The methodology uses a formal procedure for deriving Event Flow Graph from inputs. The following list shows a procedure for converting the input to EFG.

1. The Event flow to be traced from the specification diagram [12].
2. Event trace procedure in specification diagram must be started which may or may not come to end.
3. From the starting state, define a sequence of event/condition with timing constraint.
4. Highlight the visited node and arrows.
5. Repeat steps 3 and 4 until all states have been visited.

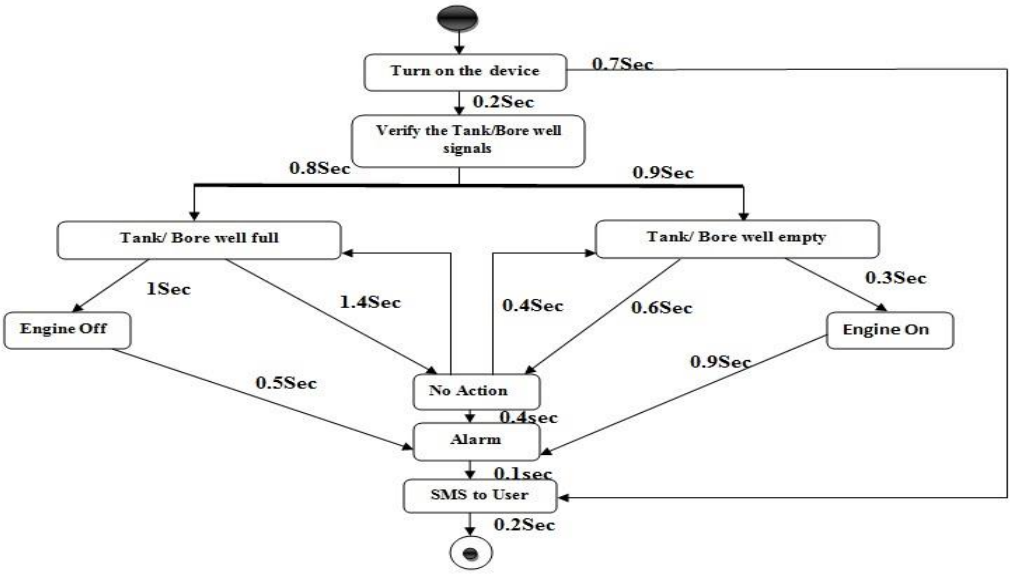


Figure.3 RTS automated water system (activity diagram)

The above Figure 3 activity diagram describes the automated water pumping system with several components consisting of water level sensor, GSM modem, PLC, and database. Sensor nodes perform the specific task, sensing and transmitting data to the end device via an inverter. Tank water level and bore water level are displayed in PC with the help of the program written in C sharp language, SMS is sent to owner's mobile. The data is saved and stored in a database which is used to build water pumping through a long-term monitoring and analysis. Consider this activity diagram for the conversion of the event flow graph by using proposed approach.

3.2 Timing event flow graph

The proposed framework TIOETG user interface receives the design specification. To find a possible interaction and components of the specification in hardware- software partitioning, the event flow graph G (N, E) is to be produced so as to minimize the complexity for automated test case generation. Moreover, in the timing event flow graph structure the relationship between the node and edge provides the flexibility for test case generation.

Algorithm 1: Convert the EFG to tree structure.

```
Pre        Contains transaction time of each node
Post       Allocated timing tree or error return
Return     TEFG generated or null
     /* Initialization */
1.  for every graph G (N,E) do
2.  G (N) = 1          /*set 1..n from initial
state . .                    to end     state of the
graph */
3.  end for;
4.  if (root node = = 0)     //Initial stage value
5.  TEFG= = null;
6.  Begin
7.  EFG [t];   // t is the transaction time
8.  If the sub node is empty then
9.  Number of node = =1;
10. Else
11.    Sub node  [t] = = EFG (N);  /* Identify
the .                                transaction time
*/
12.      If (E(LN1) < E(RN1)
13.   TEFG [n-1] = = E(LN1);
14. Else
15.      TEFG [n-1] = = E(RN1);
16. End if;
17. End if;
18. End.
```

The construction of TEFG is based on the specification input and timing sequence of the event. While extracting the specification, the properties and timing aspects of the event plays a vital role in organizing specification in order of priority. The algorithm 1 provides support for processing the input specification as a timed event flow graph (TEG) which is used for generating the test case directly.

The activity diagram for smart water system has converted as an EFG with self-loop of each node. Figure 4(A) shows the EFG for automated water system activity diagram for RTS smart water system. The EFG has to be converted to TEFG based on the timing priority. Figure 4(B) shows the TEFG based on the proposed algorithm 1

3.3 Test case generator

The presented framework converts the tree structure to stack stimulus for generating the unique test case. A stack is a linear list, all the operations to be done with restricted to one end called the stack top. For reallocating the stack or reversing the stack attributes to be performed using LIFO operation [12,13]. The stack attributes represent event information for specification diagram. Table 2 shows the EFT for activity diagram of automated water system. The number of stacks will be decided based on the linking dependence in ELT. The stack stimulus conversion of TEFG contains following steps:

- Develop the Event flow Table (EFT).
- Deriving the Event linking table with stack ID.
- Convert Event flow stack.

Table 2. Event flow table with timing priority

Stack	Event	Timing
E3	Sensor	0.5Sec
E1	Device	0.7Sec
E6	Engine	1.2Sec
E2	Status	2 sec
E10	SMS	2.4 Sec
E1	Device	2.4Sec
E5	Engine ON	3Sec
E8	Alarm	3.3Sec
E7	Alarm ON	3.5Sec
E9	Send SMS	4Sec
E4	Check status	4.5Sce

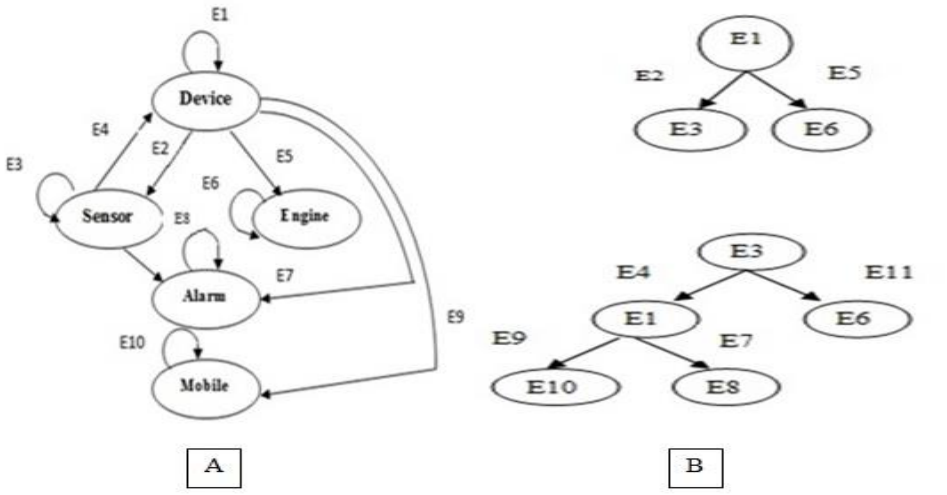


Figure.4 (a) EFG with the self-loop event, (b) Event flow tree

Based on the proposed methodology, the event flow table is developed from activity diagram. In order to minimize the flow table, boundary testing is carried out as illustrated in Event linking table i.e. Table 3. It also identifies the limit to test the data for generation of the test case. In the ELT, each stack is created based on a transaction between the events; and stack based stimulus path is used for avoiding duplicate and null values.

Test case generator is the next tool in the TIOETG framework. The tool automatically generates test cases based on the stack top. The TEFG with priority sequence to be the input but, before reaching the test case generation tool, the EFG is converted to stack based on the number of dependencies in the graph.

Table 3. Event linking table with stack ID

ID	Stimulus path
E_{T4}	E2→E5→E9→E7→E4→E11
E_{T4}	E5→E6→E7→E8→E4→E11
E_{T5}	E5→E6→E7→E8→E11
E_{T6}	E5→E6→E7→E8→E9→E11
E_{T7}	E5→E6→E7→E9→E10→E4→E11
E_{T8}	E2→E5→E9→E7→E8→E10→E1→E2→E3→E4→E11

Table 4. Generated test case for RTS

Stack Top	Test Scenario	Test case ID	Precondition(F_i)	Input(D)	Output(F_0)	Post Condition
E2	Check Status	Test Case1	Check Water level	Range(0-50)	Level(<50)	Engine start
E5	Engine ON	Test Case2	Engine power	Control	Engine on	Send message
E9	Send SMS	Test Case3	Send message	On message	User	Arm start
E7	Alarm ON	Test Case4	Arm signal	Input trigger	Sound	Check status
E4	Check status	Test Case5	Check water level	Range (0-50)	Level(>=49)	Engine stop
E11	Engine OFF	Test Case6	Engine power	Control	Off	Waiting signal

Starting from stack top, each stimulus value will produce the precondition, input value, output value and post condition. Table 4 shows the generated test case for the smart water system. The detailed explanation of test case generation for RTS along with a comparison of results is further elaborated in the experiment section. The TIOETG is demonstrated with three different specifications diagram such as sequence diagram for robot elevators lift, collaboration diagram for tours reserving and activity diagram for automated water pumping. The nature of generated test cases using proposed approach is highly suitable for RTS software. Figure 5 shows the user interface for TIOETG which has been implemented using VB.NET. The detailed comparison of test case generation for different RTS along with the influence of TIOETG is further elaborated in the next section.

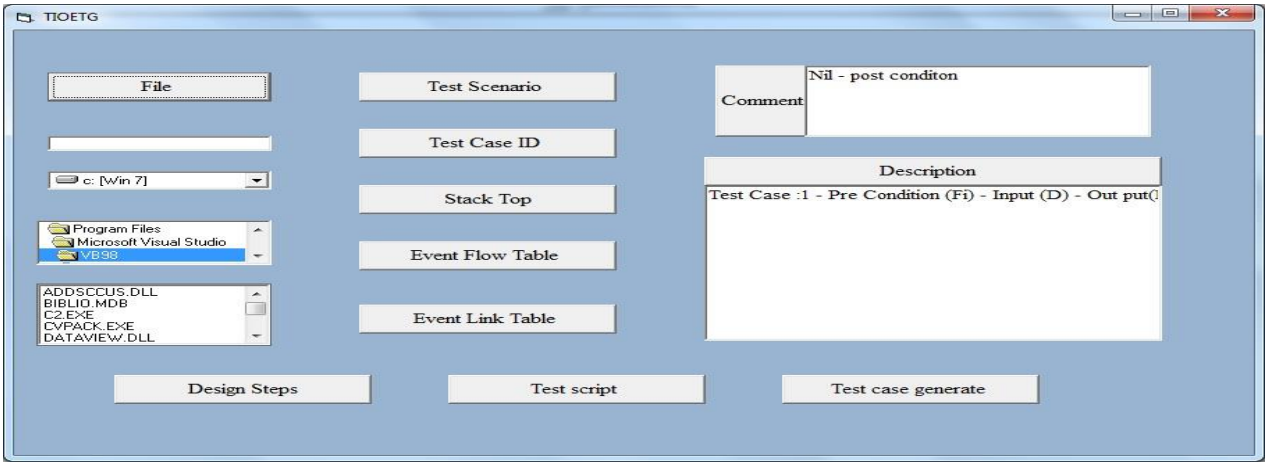


Figure.5 The interface of TOETG test case generator

4. Experiments and Results

The experiments were performed to determine whether the TIOETG improves the quality of RTS application and generate the possible test cases. This paper compares the performance of TIOETG with three different RTS specifications. Thereby, one of the experiments is to be compared with the existing approach that attempts to determine the possible test cases and coverage criteria.

4.1 Client –server based RTS

The robot is designed to help in lifting the car from one place so as to place it on a truck. An RTS application will be designed to receive information about the object that is to be lifted. The object can either be a car or a block of metal. If it is a car, the robot arm will automatically activate a gripper. If the object is a black of metal other than the car, the robot arm activates a motor that controls an iron rope [15].

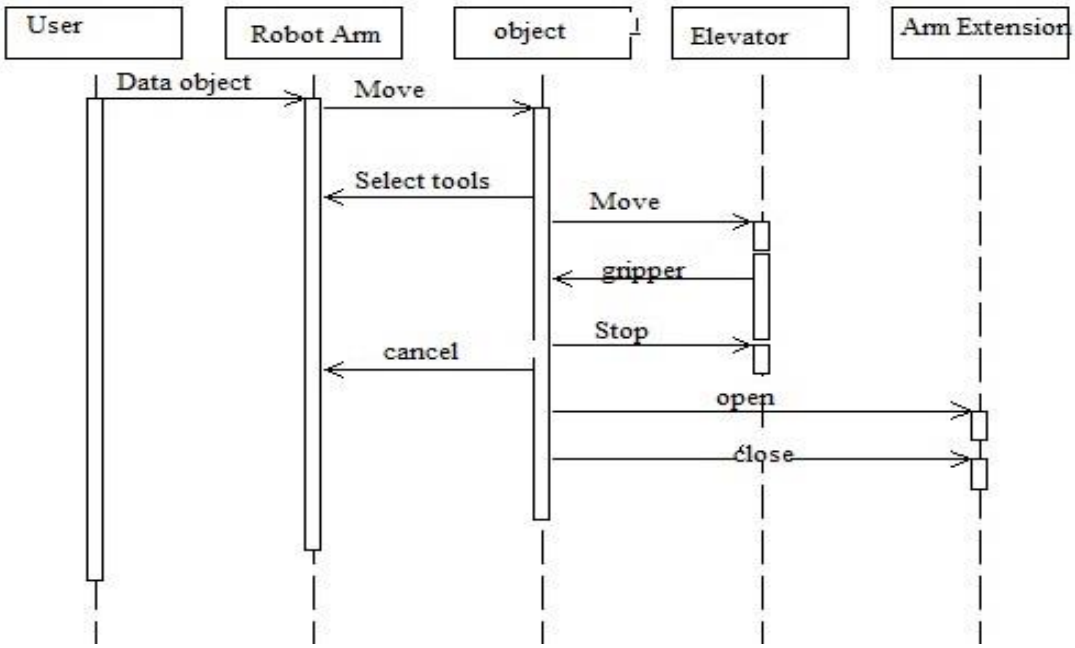


Figure.6 Robot elevators lift (object sequence)

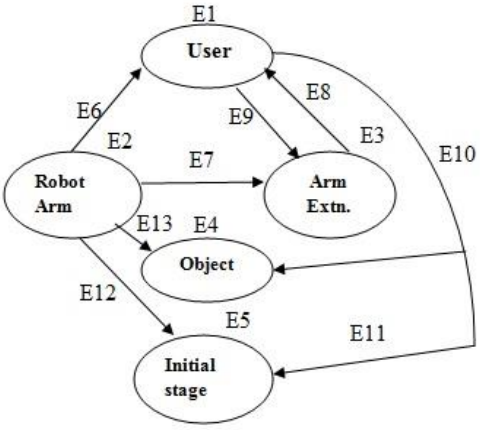


Figure.7 EFG for robot elevators lift

UML sequence diagram among the objects in a client –server based RTS is shown in Figure. 6. According to the proposed framework, SLT developed for robot elevator lift. Figure 7 shows EFG. Finally, a generated test case for robot elevator lift is shown in Table 5.

4.2 Embedded system based RTS

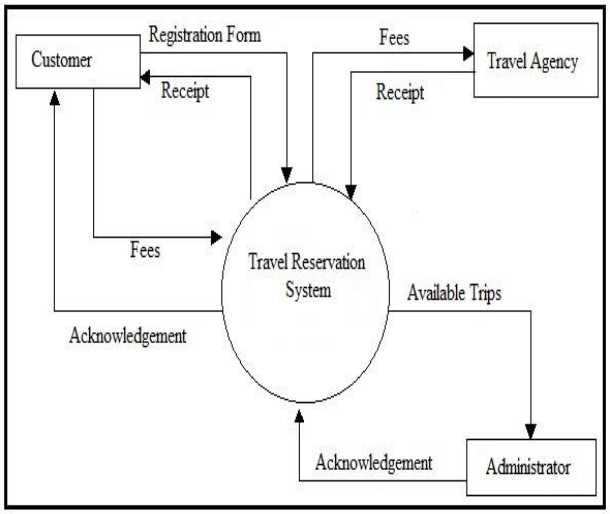


Figure.8 Tours reserving system (collaboration diagram)

Figure.8 shows a software system for a travel agency provides reservation facilities for the people who wish to go on tours by accessing a built-in network at the agency. The application software keeps information on tours. The user can access the system to make a reservation and cancel a reservation. Any complaints or suggestions that the client may have could be sent by e-mail to the agency or stored in a complaint database. Employees could add, delete and update the information on the customers and the tours. For security purposes, the employee should be provided a login ID and password. This system is the combination of input control, an S3C210 processor with embedded platform, communication network and output smart mobile [15]. The below Figure 9 shows the percentage of test cases based on node, event, and self-loop using the proposed approach.

4.3 Comparative Results

The proposed framework provides the possible test cases with triplet value (Fi, D, Fo).The presented technique actives 99% statement coverage and 98% of functional coverage by implementing the embedded based smart home RTS [12]. The performance of our proposed approach compared with the existing approach based on intermediate model of graph and tree. The results show that stack based approach have better performance than the previous approach. Figure.10 shows the comparative results of existing and proposed test case generation approach for smart home system.

Identification of stimulus process requires more efforts and time in case if it is manual and also there is more possibility of not getting the desired results. Therefore, the proposed TIOETG framework provides the possible test cases and coverage criteria such as all paths, all nodes, and all transaction for RTS. In summary, it has been noticing the following about TIOETG framework with practices of three RTS. TIOETG framework provides the possible test cases based on the selection flow graph. Integration of event flows in the stack is suitable for large systems. Test analyses ensure the event coverage. The tool components such as graph converter, test case generator and algorithm for TEFG converter can be customized and easily extended with specification diagram.

Table 5. Generated test cases for RTS

Stack Top	Test Scenario	Test case ID	Precondition(F_i)	Input(D)	Output(F_0)	Post Condition
E1	Object details	Test Case1	Analyze data	Select tool	Move Arm	Fixes base
E2	Robot Arm	Test Case2	Power	Control	Engine on	Send message
E3	Arm Extension	Test Case3	Send message	On message	User	Arm start
E4	Object	Test Case4	Arm signal	Input trigger	Sound	Check status
E6	Check status	Test Case5	Check object	Move	Move left	Arm stop
E11	Send signal	Test Case6	Arm power	Control	Off	Waiting signal
E5	Initial stage	Test Case7	Stop signal	Initial stage	Analyze data	Select tool

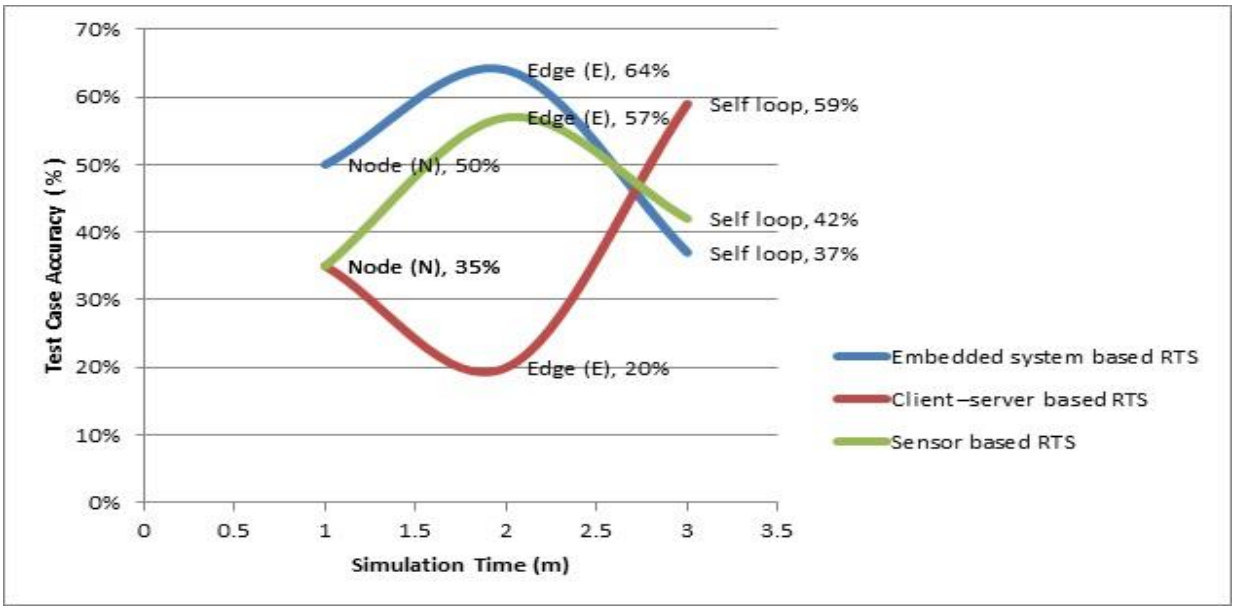


Figure.9 Generated test cases using TIOETG framework

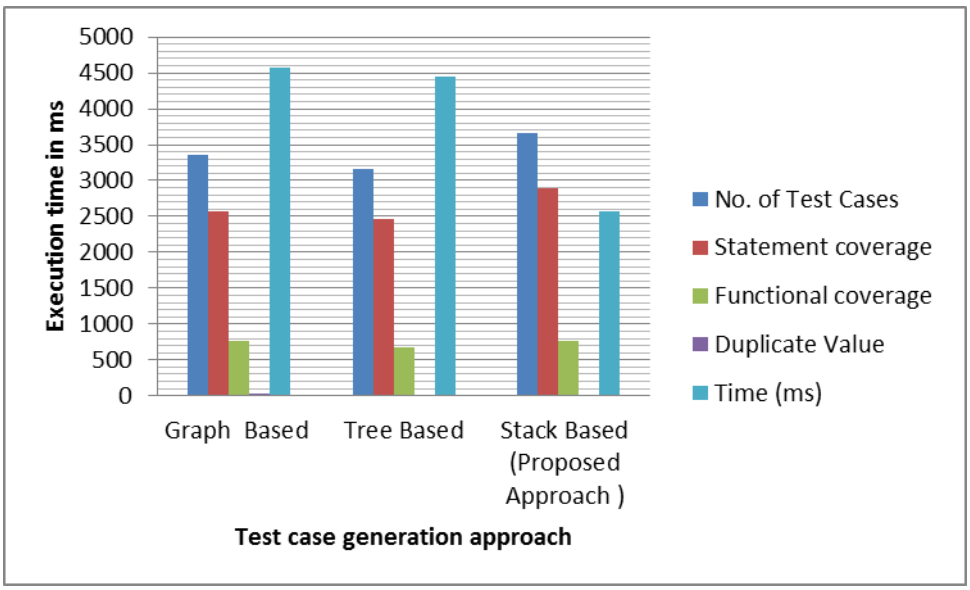


Figure.10 Performance of TIOETG

5. Conclusion

The presented framework with the event flow graph converter tool for the generation of test cases yields efficient test cases for real - time systems. If the testers do not fully understand this event flows then the problem is compounded in RTS testing. To address these challenges, the proposed approach converts the EFG with timing sequence. The timing event flow graphs are to be formed based on the timing of node and sub-node. The presented

algorithm converts the EFG to TEFG based on the tree traversal. The methods for test case generation are based on the stack operation. So, it prevents duplicate and null values with 99% of path coverage. The generated test cases cover the post condition using stack based methods. The experimental work for three RTS has shown possible test cases. The generated test cases provide new faults that have not been detected in the previous approach and cover all paths for RTS. However, there remains a probability that a poorly chosen set of events will yield a test case that does not provide possible coverage. In future, the methodology could be tried with other software specification diagrams for covering all the possible event transaction.

References

[1] W. L. Andrade and P. D. L. Machado, "Generating Test Cases for Real-Time Systems Based on Symbolic Models", IEEE *Transactions on Software Engineering*, Vol. 39, No. 9, pp. 1216–1229, 2013.

[2] M. S. AbouTrab, M. Brockway, S. Counsell, and R. M. Hierons, "Testing Real-Time Embedded Systems using Timed Automata based approaches", *Journal of Systems and Software*, Vol. 86, No. 5, pp. 1209–1223, 2013.

[3] Y.C. Wu and C.F. Fan, "Automatic test case generation for structural testing of function block diagrams", *Information and Software Technology*, Vol. 56, No. 10, pp. 1360–1376, 2014.

[4] P. Mani and M. Prasanna, "A study on functional specification based test case generation for real-time systems", *International Journal of Engineering and Technology*, Vol. 8, No. 4, pp. 1801–1806, 2016.

[5] S. K. Swain, D. P. Mohapatra, and R. Mall, "Test Case Generation Based on State and Activity Models", *Journal of Object Technology* Vol.9, No.5 pp.1-27, 2010.

[6] M. Chen, P. Mishra, and D. Kalita, "Efficient test case generation for validation of UML activity diagrams", *Design Automation for Embedded Systems*, Vol. 14, No. 2, pp. 105–130, 2010.

[7] P. Samuel, R. Mall, and P. Kanth, "Automatic test case generation from UML communication diagrams", Information and Software Technology, Vol. 49, No. 2, pp. 158–171, 2007.

[8] E. Jee, D. Shin, S. Cha, J.-S. Lee, and D.-H. Bae, "Automated test case generation for FBD programs implementing reactor protection system software: Automated Test Case Generation For FBD Programs", *Software Testing, Verification and Reliability*, Vol. 24, No. 8, pp. 608–628, 2014.

[9] S. Fujiwara, K. Munakata, Y. Maeda, A. Katayama, and T. Uehara, "Test data generation for web application using a UML class diagram with OCL constraints", *Innovations in Systems and Software Engineering*, Vol. 7, No. 4, pp. 275–282, 2011.

[10] E. P. Enoiu, D. Sundmark, and P. Pettersson, "Model-Based Test Suite Generation for Function Block Diagrams Using the UPPAAL Model Checker", *In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pp.158–167, 2013.

[11] P. Mani and M. Prasanna, "Automatic test case generation for programmable logic controller using function block diagram", *In IEEE International Conference on Information Communication and Embedded Systems* (ICICES), pp. 1–4, 2016.

[12] The blog for event trace, *available from November 2016*: http://rtstestcase.blogspot.in/

[13] R. F. Gilberg and B. A. Forouzan,"Data Structures: A Pseudocode Approach with C", 2nd ed., *TATA McGraw Hill Edition, India*, 2005.

[14] J. Ghinwal , "UML by Example", First ed., *Cambridge universtiy press*, UK , 2004.

[15] A. Nayak and D. Samanta, "Automatic Test Data Synthesis using UML Sequence Diagrams", *Journal of Object Technology*, Vol. 9, No.2, pp.115-144, 2010.

[16] H. Wang, J. Xing, Q. Yang, W. Song, and X. Zhang, "Generating effective test cases based on satisfiability modulo theory solvers for service-oriented workflow applications: Effective Test Cases For Service-Oriented Workflow Applications", *Software Testing, Verification and Reliability*, Vol. 26, No. 2, pp. 149–169, 2016.