Journal of Engineering Science and Technology
Vol. 12, No. 4 (2017) 860 - 874
© School of Engineering, Taylor's University

TEST CASE GENERATION FOR EMBEDDED SYSTEM SOFTWARE USING UML INTERACTION DIAGRAM

MANI P., PRASANNA M.*

School of Information Technology and Engineering,
VIT University, Vellore, Tamil Nadu, India
*Corresponding Author: prasanna.m@vit.ac.in

Abstract

Software development process contains various phases. More efforts and cost have to be spent in the testing phases. Test case generation at cluster level in Software Development Life Cycle (SDLC) can be the best optimised solution for reducing effort and cost. The efficient test cases will play a vital role in reducing the effort in Software Testing Life Cycle (STLC). Unified Modeling Language (UML) designs provide valid information for software development process. UML interaction diagram based test case generation can be used to improve the quality in software. This paper presents a method for test case generation from UML interaction diagram at the cluster level. It makes three major processes. First, interaction diagrams are converted to data structure stack, and then the stack stimulus are minimized using boundary testing, and finally the test case is generated from minimized stack. This paper has presented our technique with some real time examples of embedded system software.

Keywords: Test case generation, Software testing, Stack approach.

1. Introduction

The development of embedded system is an important activity in the digital world. Producing high quality software in the real time activity is becoming a major problem in embedded system design due to the complexity of software coding and testing. Effective testing of software is necessary to produce reliable systems [1]. Effective testing has been a major bottleneck in the software development process. In most software development project, more than 50% of developments efforts are typically spent on testing [2]. The efforts of the software developer and cost can be reduced if the process of testing is automated fully. Moreover, test case plays an important role in the process of automatic testing.

Nomenclatures

D	Stack data
F_{in}	Input function
F_{out}	Output function
F_S	Minimized stack
F_X	Stack top
M_n	Message sequence
S_n	Stack sequence

Abbreviations

EMF	Eclipse Modeling Framework
IF	Intermediate Format
LIFO	Last In First Out
OCL	Object Constraint Language
SDLC	Software Development Life Cycle
SLT	Stimulus Linking Table
STLC	Software Testing Life Cycle
UML	Unified Modeling Language
UTG	UML behavioural Test case Generator

Most directed test case generation work is performed by human intervention. Hand - Written test cases entail labourers and time - consuming effort of verification engineers who have deep knowledge of the design under verification. Due to the manual development, it is difficult to generate all directed test cases to achieve a coverage goal [2]. Automatic test case generation is the alternative solution for this problem. In the software development process, testing can be divided into four levels i.e. algorithmic level, class level, cluster level and system level [3]. Figure 1 shows the levels of testing .The algorithmic and class level test will test the software in the basic program. System level testing is performed to test the whole system (including cluster level). The cluster level testing is used to test the classes in the stage of design.

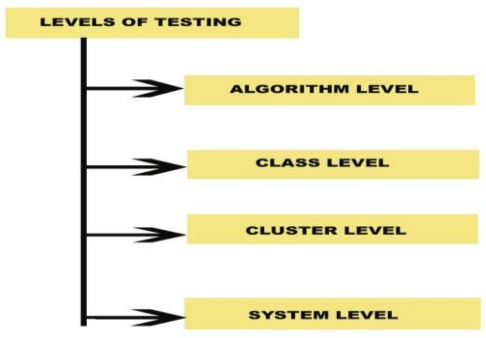


Fig. 1. Levels of testing.

Test case generation from design specifications has the added advantage of allowing test cases to be available early in the development cycle [3]. UML interaction diagrams are suitable source of information for test case generation.

UML is becoming a promising specification language for both software and hardware designs [4]. UML supports different types of interaction diagrams including sequence and collaboration diagrams [5]. In this research work, UML based test case generation process is proposed to reduce the testing effort and to improve the quality of the software.

This paper presents a method for generating test cases based on UML interaction diagram. Section 2 presents the related work on UML based test case generation. Section 3 has a brief discussion on test case generation and definitions, which have been used in the said approach. Section 4 proposes our methods for test case generation. A detailed illustrations using real time embedded and results constitute the Section 5. Finally, the conclusions are provided in Section 6.

2. Related work

This section provides related research in the area of UML based test case generation. Almost all UML diagrams are positively used in embedded system design [1]. Cavarra et al. [2] proposed methods to translate Intermediate Format (IF) from UML diagrams. The IF describes the behaviour of a system. They also provide IF based test case generation scheme. Abdurazik et al. [4] has described novel test condition based test case generation for collaboration diagram. They have used a traditional data flow criteria. Fujiwara et al. [5] has given the formal description of web application behaviours and data constrains with Eclipse Modeling Framework (EMF) class diagrams and Object Constraint Language (OCL) notation. They also have introduced test data generation scheme based on formal model. The intermediate model of the related research are summarised in Table 1.

Table 1. Test case generation from UML.

Author	Input Model	Method	Intermediate Model	Coverage Criteria
Cavarra et al., 2004 [2]	Class Diagram State Diagram Object Diagram	Traversal, Function minimization	Formal behavioural description	Test graph All traces paths
Abdurazik and Offutt 2000 [4]	State Diagram	BFS Traversal	State transition table	Class level testing
Kansomkeat et al., 2010 [6]	Activity Diagram	Bottom –up Testing Strategy	Condition classification tree	Class, Methods
Li et al., 2007 [7]	Sequence Diagram	DFS, Category partition	Scenario tree	Message, path, Conditions
Nayak and Samanth 2010 [8]	Sequence Diagram	DFS traversal	Structured control graph	All path and Condition

Samuel and Mall [9] has presented methods for generating testable sequence diagram. They have proposed a test tool for generating automated test stubs based on testable sequence diagram from behaviour specification. They have generated corresponding test case data sets. Bell and Haverkort [10] describes the lost effectiveness methods of a transition tree, using a BFS traversal using UML state machine diagram. Samuel et al. [11] has proposed a method to generate test cases from UML state diagram by using selected predicate boundary value analysis. They also implemented UML behavioural Test case Generator (UTG) for generating test case from state diagram.

The above mentioned research works used automatic and manual methods to develop test cases from one primary UML diagram. Apart from this, these approaches also make use of some other UML diagrams to obtain additional information for generating test path. As a result of which, the number of path and statements increases, thereby leaves scopes for more number of faults. Detection of such faults in all combinations is a major problem. If software requirements are to be modified, the Software Development Life Cycle needs to be changed. Such modifications also require re-testing of software. Hence, our proposed methods i.e. stack-based approach aims at giving solutions by making changes in the software.

3. Test case generation using stack

3.1. UML

UML is popularly used for interaction level specification language in embedded system design. To avoid the complexity of embedded system and improve the quality of system program, interaction diagrams are important in the UML. An interaction is a unit of behaviour that focuses on the observable exchange of information between the objects [12]. An interaction diagram describes the structural relationship between the objects. This diagram shows the communication through a structural view of the system. In UML interaction diagram, interaction line or arcs between the objects will describe the messages and their sequences. The widely used interaction diagrams are sequence diagrams and collaboration diagrams. This paper describes the basics and formal definitions of UML interaction diagram and how the interaction diagram helps in the process of developing the test cases.

3.2. Sequence diagram

A sequence diagram represents the relationships among objects through inter-object links. This diagram is dynamic model that supports a dynamic view of the evolving systems. It shows the explicit sequence of passed messages between objects in a defined interaction [13].

3.3. Test case

According to the International Software Testing Qualifications Board (ISTQB) definition test case is a set of input values, execution preconditions, expected results and execution post conditions, developed for a particular objective or test condition, such as exercising a particular program path or verifying compliance with a specific requirement [14].

Test cases are commonly designed based on program source code [15]. A test case has triplet value [F_i D F_o]: *Fi* is the initial status of the system and acts as a function input to initiate the process; *D* indicates the process of retrieving the test data, and F_o is the expected output of the system.

Figure 2 shows the test case generation methods. The final outcome is producing the test case from F_x.

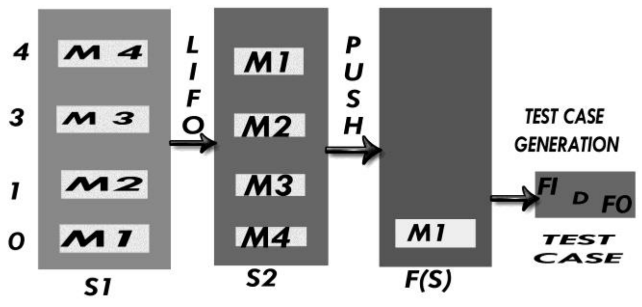


Fig. 2. Test case generation from stack.

3.4. Stack operation

Data structure stack can be represented using an array. All the stack operations will be performed at the single end. Based on different primitive operations on stack we can create, add and remove the stack elements [16]. Each stack can be divided into two parts, one for storing the actual data and the other for marking the position of the stack elements. Figure 2 shows the stack elements *M1, M, M3 ..*M_n are the actual data and outside the stack *0, 1, 2...n* represents the position of the stack elements. The methods of inserting new elements into the stack are called as push operation. If the stack is already full or in other words lacks the sufficient space to add the new element, then it is known as stack overflow. The procedure of removing data from the stack is called as operations of pop. If we attempt to remove an element from already empty stack then it will lead to stack underflow condition.

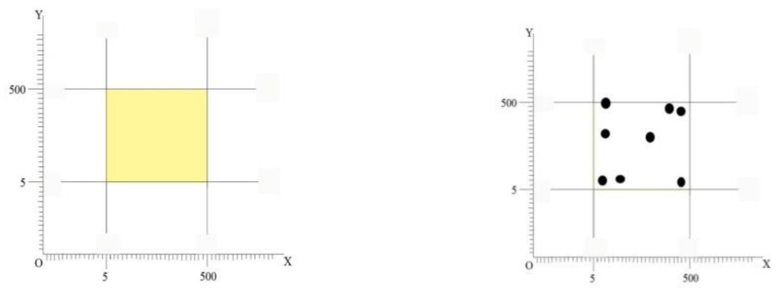
3.5. Stimulus path

A stimulus path is a sequence of data transfer from one lifeline to another in interaction diagram. Those stimulus paths are to be converted as a stack data as per our proposed methods. The stack data collected from sequence diagram are represented with different stack names.

3.6. Boundary value analysis

Boundary value analysis and equivalence partitioning both are test case analysis strategies. A boundary might consist of several segments and each segment of the boundary is called a border. Each border is determined by a single simple predicate in the path condition [17]. Boundary value analysis determines the top and bottom level of the test case limitation. The boundary value typically is used to reduce the total number of test case.

Consider an example with two input variables X and Y which have been specified by intervals of 5 & 500 respectively. If the boundary value condition of X axis and Y axis is fixed at above 5 and below the 500, it covers all the possible test cases between 5 & 500 as the boundary stimulus values are 5, 6,499 and 500. This has been shown in Fig. 3. As per the proposed methods, it can be observed that the boundary coverage can identify the requirement based stimulus from stack easily. This coverage produces test cases which have more efficiency in detecting errors. Therefore, if the boundary value analysis is not carried out, then there remains a possibility that the same sequence messages are validated again and again by conducting a number of test cases to obtain the desired results.



(a) Boundary coverage **(b) Boundary value analysis**

Fig. 3. Boundary coverage for two values.

4. Proposed test case generation method

This section briefly describes our methods to generate the test cases using UML interaction diagram. Its flow represents the test case generation using Stimulus Linking Table (SLT). First our approach is to covert the stack array from interaction diagram. In the next step, using Last In First Out (LIFO) select the data item from origin stack to new empty stack. Then the stacks are minimized as a data item using boundary testing method. In the final step, test cases are generated from the minimized stack data. This process is repeated until the stack is empty. The steps in test case generation methods are neatly explained in detail with real time embedded system example in the following subsections.

The sequence diagram contains two elements i.e. header and body. The header section represents the components of the system whereas the body section describes the communication information between the headers. Sequence diagrams are organized according to transaction time between the objects. Figure 4 shows a sequence diagram model for communication among objects. The dotted lines from top to bottom are lifeline of the objects. It can increase as long as the object exists.

The objects that send a message to other objects are shown by an arrow between their lifelines [14]. In our proposed work, these messages are converted to stack for test case generation.

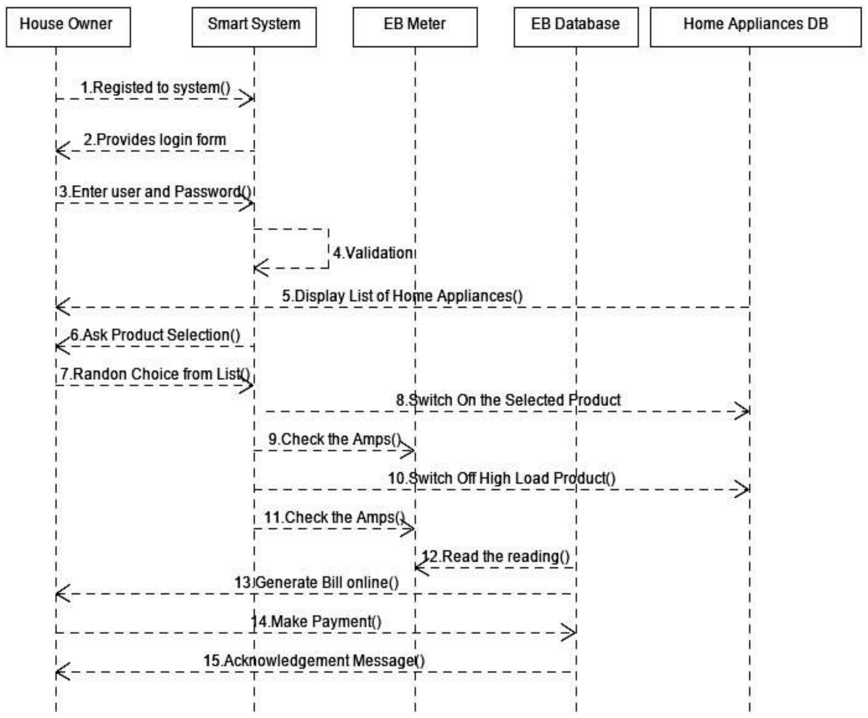

House Owner
Smart System
EB Meter
EB Database
Home Appliances DB
1.Registed to system()
2.Provides login form
3.Enter user and Password()
4.Validation
5.Display List of Home Appliances()
6.Ask Product Selection()
7.Randon Choice from List()
8.Switch On the Selected Product
9.Check the Amps()
10.Switch Off High Load Product()
11.Check the Amps()
12.Read the reading()
13.Generate Bill online()
14.Make Payment()
15.Acknowledgement Message()

Fig. 4. Smart home EB systems (sequence diagram).

These UML sequence diagrams illustrate the process of automated smart home EB system for test case generation. This sequence diagram contains object for various communication messages. With a set of microcontrollers with embedded codding, the system works for validation of user and product selection. The simulated coding also checks the Amps in EB meter based on the condition fixed in embedded program. Then this process automatically generates the EB bill and amount is debited from the account of the house owner.

4.1. Stack conversion

A stack is an abstract data type that serves as a collection of elements with two principal operations (PUSH, POP) [16]. Figure 5 shows the methods for generation of test cases from interaction diagram. 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.

The stack conversion of sequence diagram contains following steps:

- Draw Sequence graph.
- Develop the SLT.
- Deriving the Stimulus path.
- Convert SLT stack.

The Sequence diagram is converted as number of stack based on the stack dependence. The size of the stack needs to be identified based on sequence graph nodes. Figure 6 shows the sequence graph for validation process.

The stack attributes represents messages in sequence diagram. The number of stacks will be decided based on the linking dependence in SLT. SLT will have the following columns. Table 2 describes the SLT for given sequence diagram.

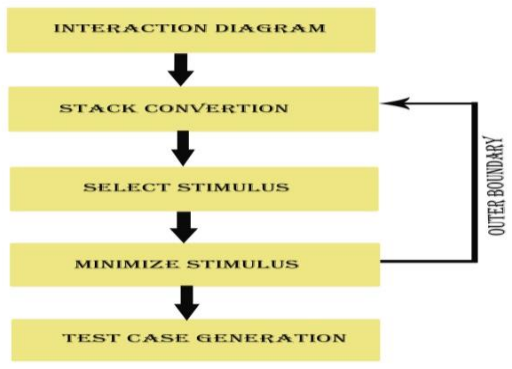


Fig. 5. Test case generation process.

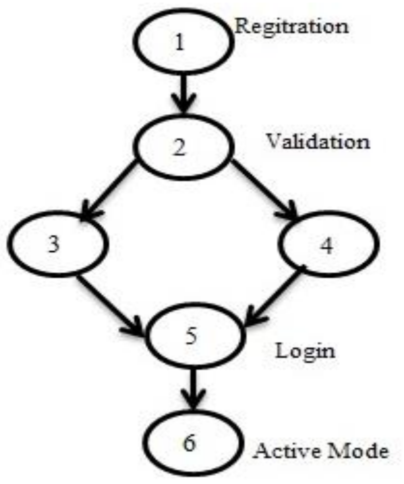


Fig. 6. Sequence graph.

a) **Stack Name**- It is an alphanumerical value used to be describe the stack number.
b) **Message**- Describe the stack attributes or message between the stacks.
c) **Sequence number**- Sequence number will represent the flow of message number.
d) **Linking stack**- This field describes the dependency of each message.

From the SLT, the stimulus data are collected along with the information regarding the interaction among the stimulus.

Table 2. SLT for smart home EB system.

Stack Name	Message	Sequence number	Linking stack
M1	New registration	1	-
M2	Login	2	1
M3	Validation	4	2,16
M4	List display	5	4
M5	Product Request	6	5
M6	Product selection	7	6
M7	Switch on product	8	7
M8	Check Amps	9	8,16
M9	Switch off product	10	9
M10	Check Amps and reading	12	8,10,16
M11	Generate bill	13	12
M12	Make payment	14	13
M13	Return confirmation	15	14
M14	End process	16	4,9,12,15

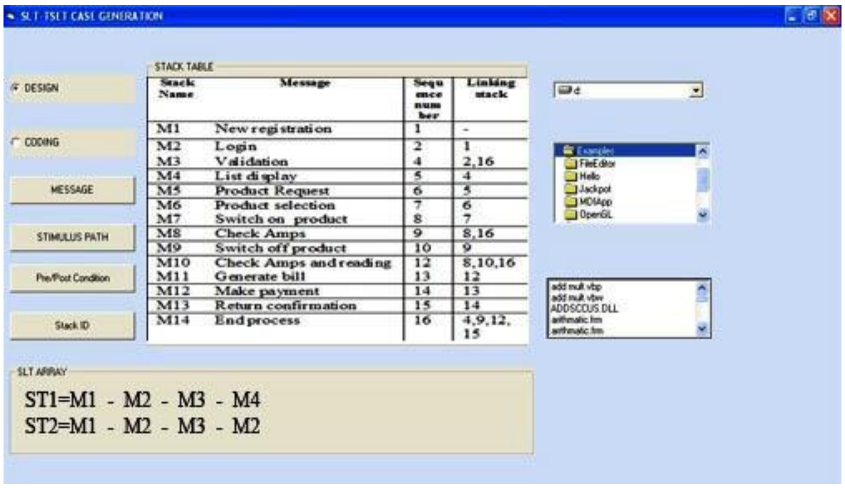
Algorithm 1. SLT stack array conversion

```
Pre       $Stack_{E1}$ contains size of stack
Post      SLTarray allocated or error returned
Return    SLTarray head or null if no memory
if (memory not available)                  //initial stage (or) overflow
        SLTarray == null
else
        allocate (stimulus)        = 0
        SLTarray → count           = 0
        SLTarray→ top              = -1
        SLTarray →stackMax         = $stack_{E1}$
end if
if (memory not available)                  //During allocation of the new stack.
        recycle (SLTarray)
        SLTarray = null
elseif(SLTarray stack depends single stimulus )
        allocate (sltarray→stimulus)
else
        SLTarray++
        reallocate(sltarray→ stimulus)
return SLTarray                 // Stack underflow
end create SLT stack.
```

In the next process of stack conversion, each message is converted as a stack array. If stack message interacts with more than one stack then reallocate takes place for the new stack array. The above algorithm SLT stack array conversion explains in detail about allocation of stack array.

According to the algorithm 1, first check the memory space before allocating the elements to SLT array. If the stack is over flow then return null value or else it allocates new stack array.

The elements are allocated to *SLTarray1*, *SLTarray2….SLTarrayN*. The number of array stack is increased based on dependence of each stimulus in Stimulus dependence table. Messages are assigned the value from M_1 *to* M_n and also each message has a unique number and no duplicate value. While creating stack array from sequence diagram, if the stack is empty and if there is no communication between the life lines, then the stack is to be destroyed. According to the above algorithm Table 3 will be generated with stack ID named ST_3 *to* ST_8. Figure 7 shows the user interface for *SLTarray* generation which has been implemented using VB.NET.



Stack Name	Message	Sequence number	Linking stack
M1	New registration	1	-
M2	Login	2	1
M3	Validation	4	2,16
M4	List display	5	4
M5	Product Request	6	5
M6	Product selection	7	6
M7	Switch on product	8	7
M8	Check Amps	9	8,16
M9	Switch off product	10	9
M10	Check Amps and reading	12	8,10,16
M11	Generate bill	13	12
M12	Make payment	14	13
M13	Return confirmation	15	14
M14	End process	16	4,9,12, 15

ST1=M1 - M2 - M3 - M4
ST2=M1 - M2 - M3 - M2

Fig. 7. Interface of SLT test case generator.

4.2. Minimize the stimulus

Based on the proposed methods, the stack array is developed from sequence diagram. In order to minimize the stack stimulus, boundary testing is carried out as illustrated in Table 3. It also identifies the limit to test the data for generation of test case. In the SLT Array each stack is created based on messages between the objects, the upper bound of which needs to be fixed according to number of node in sequence graph. The selection of stack is used for avoiding duplicate and null values.

In the SLT array, assume two points named YES and NO for a given boundary satisfying the boundary testing criterion.

Table 3. SLT array conversion.

Stack ID	Stimulus path
S_{T3}	M5→M6→M7→M8
S_{T4}	M5→M6→M7→M8→M14
S_{T5}	M5→M6→M7→M8→M10
S_{T6}	M5→M6→M7→M8→M9→M10
S_{T7}	M5→M6→M7→M9→M10→M14
S_{T8}	M5→M6→M7→M8→M10→M11→M12→M13→M14

We transform the relational expressions to the top stack. The expression constructed in the form of ‘condition 1 relational operator condition2’, here condition1 and condition2 are arithmetic expressions of boundary value. For example stack top =(live voltage >=600), in the smart home EB system consider live voltage 500 then the boundary testing criterion return the value NO, then using Pop operation transfer the stimulus to another stack , the same step has to be performed until the stack becomes empty. During this process we can cover the entire stimulus in the stack at least once.

4.3. Generate test case

Starting from stack top, each stimulus value will produce the precondition, input value, output value and post condition. F_{in} value will be inside the boundary value and F_{out} will be outside the boundary. These two points have different opposite values in the graph and pushed into the stack. These values are used for generating test case in the corresponding stack stimulus. Table 4 shows the test case description for stack ID ST_4. This generated test case identifies the entire test path for smart home EB system based on the sequence diagram. The proposed approach also covers the message path condition using stack based methods. The detailed explanation of test case generation for smart home security system along with comparison of results is further elaborated in the section 5.

Table 4. Test case for smart home EB system.

Stack top	Test Scenario	Test Case ID	Precondition	Input	Output	Post condition
M5	Product Request	Test Case1	Validation	Product list request	List of item in home	Display product
M6	Product selection	Test Case2	Display All Product list	Select item(X)	Block item(X)	Active mode
M7	Switch on product	Test Case3	Active mode	Product Id(X)	Switch on (X)	Check condition
M8	Check Amps	Test Case4	Get Condition	Get Amps	Check used Amps	Display total Amps
M9	Switch off product	Test Case5	Display Amps	Product ID(x)	Switch off(x)	Check Condition
M10	Check Amps and reading	Test Case6	Get condition	Get Amps(t)	Check used Amps	Display used Amps

5. Example and Results

Figure 8 shows a working principle for smart home security system. This system is the combination of input control, S3C210 processor with embedded platform, video processor; communication network and output smart mobile. When the input control is pressed, the system immediately sends the warning signal to processor. The processor controlled camera, installed above the input control, captures the image which is then transferred to embedded platform for processing the image and save as an image file format. Finally, the input image file is transferred through the communication network to targeted smart mobile.

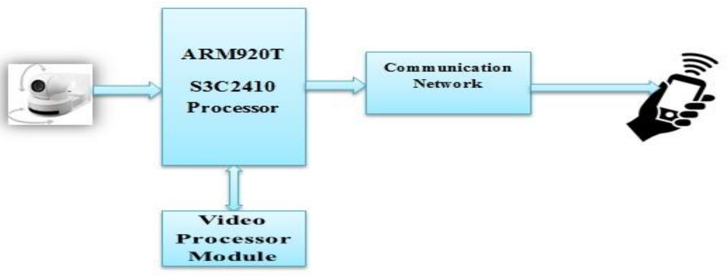


Fig. 8. Working principle of smart home security system.

The UML sequence diagram among the objects in a smart home security system is shown in Fig. 9. According to the proposed approach in section 4, Table 5 shows SLT followed by SLT Array in Table 6. Finally generated test case for smart home security system is shown in Table 7.

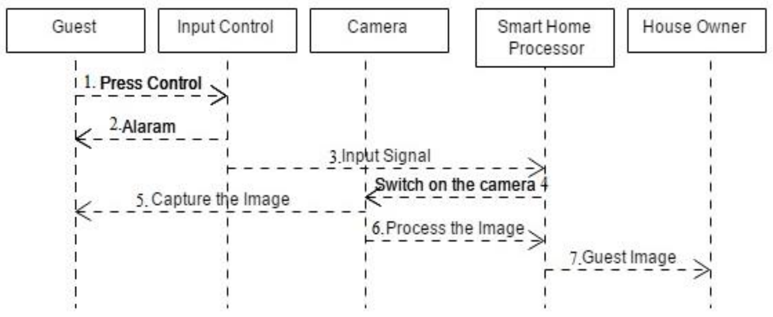


Fig. 9. Smart home security systems (sequence diagram).

Table 5. SLT for smart home security systems.

Stack Name	Message	Sequence number	Linking stack
M1	Press input control	1	-
M2	Alarm	2	1
M3	Input signal	3	1
M4	Switch on the camera	4	3
M5	Capture the image	5	4
M6	Process the image	6	5
M7	Transfer image to mobile	7	6
M8	Accept the request	8	7
M9	Reject the request	9	7
M10	Switch off the system	10	1,7,8

Table 6. SLT array conversion.

Stack ID	Stimulus path
$\mathbf{S_{T1}}$	M1→M2→M10
$\mathbf{S_{T2}}$	M1→M2→M3→M4→M5→M6→M7→M9→M10
$\mathbf{S_{T3}}$	M1→M2→M3→M4→M5→M6→M7→M8→M10

Table 7. Test case for smart home security systems.

Stack top	Test Scenario	Test Case ID	Precondition	Input	Output	Post condition
M1	Input Control	Test Case1	Fix input the control	Press the control	Alarm Input	Send the signal
M2	Alarm	Test Case2	Alarm input	Input signal(x)	Sound(x)	Active mode
M3	Input Signal	Test Case3	Active mode	Product signal	Switch on	Power signal
M4	Switch on camera	Test Case4	Get signal	Active camera	Capture image	Store the image
M5	Capture the image	Test Case5	Store image	Capture	Store image	Resize
M6	Process the image	Test Case6	Image size	Image	Reduce the image	Transfer
M7	Transfer the image	Test Case7	Reduced image	Network signal	Identify Receiver	Accept/reject
M8	Accept request	Test Case8	Refer the image	Guest image	Accept the request	Exit

The presented technique achieves 98% of statement coverage and 99% of functional coverage by implementing the embedded code for smart home systems. It also covers many test coverage i.e. object coverage, message path coverage, condition coverage.

According to the generated test case for embedded real time system, the numbers of test cases are minimized and they achieve all the path coverage. The comparisons of test coverage in the smart home EB system and smart home security system are shown in Fig. 10.

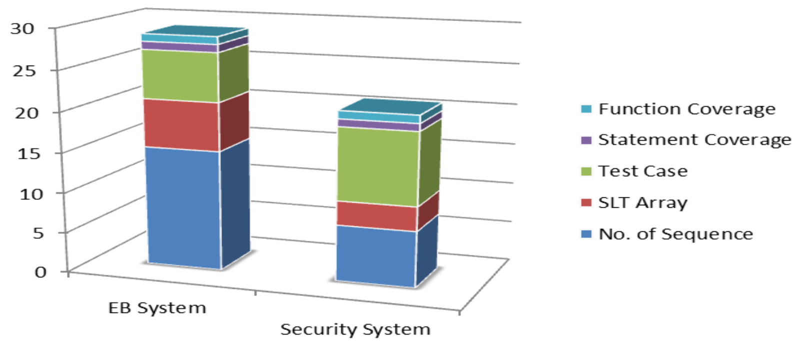


Fig. 10. Comparison of test coverage for smart home systems.

6. Conclusions

The proposed approach for generating test case from UML interaction diagram by using stack array and boundary value techniques yields efficient test cases. It also shows how to create stimulus linking table from interaction diagram. Some concluding observations from the research are given below.

- The presented algorithm is very effective for converting stack array from stimulus linking table.
- The methods for test case generation based on stack array and boundary value achieves high test coverage with message path condition.
- It has illustrated the real time applications of embedded system by using UML interaction diagram which further shows the behaviour of the system.
- The outcome of our proposed approach indicates the possible test path to test real time applications of embedded system.

In future, the functional test case generation of our proposed work could be tried with other UML diagrams to fully automate the test cases to test the real time embedded system.

References

1. Binder, R.V. (1996). Testing object oriented software a survey. *Journal of Software Testing Verification and Reliability*, 6(4), 125-252.
2. Cavarra, A.; Crichton, C.; and Davies, J. (2004). A method for the automatic generation of test suites from object models. *International Journal Software Technology,* 46(5), 309-314.
3. Samuel, P.; Mall, R.; and Bothra, A.K. (2008). Automatic test case generation using unified modelling language state diagrams. *IET Software,* 2(2), 79-93.
4. Abdurazik, A.; and Offutt, J. (2000). Using uml collaboration diagrams for static checking and test generation. *Proceedings of Advancing the Standard Third International Conference UK,* 383-395.
5. Fujiwara, S.; Munakata, K.; Maeda, Y.; Katayama, A.; and Uehara, T. (2011). Test data generation for web application using a UML class diagram with OCL constraints. *Innovations in Systems Software Engineering,* 7(4), 275-282.
6. Kansomkeat, S.; Thiket, P.; and Offutt, J. (2010). Generating test cases from uml activity diagrams using the condition-classification tree method. *Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE),* 62-66.
7. Li, B.; Li, Z.; Qing, L.; and Chen,Y. (2007). Test case automate generation from uml sequence diagrams and OCL expression. *Proceedings International Conference on Computational Intelligence and Security*, 1048-1052.
8. Nayak, A.; and Samanth, D. (2010). Automatic test data synthesis using UML sequence diagram. *Journal of Object Technology,* 9(2), 115-144.
9. Samuel, P.; and Mall, R. (2008). A novel test case design technique using dynamic slicing of UML sequence diagrams. *E-informatica Software Engineering Journal,* 2(1), 71-92.
10. Bell, A.; Haverkort, B.R. (2005). Sequential and distributed technology transfer. *International Journal on Software Tools Technology transfer,* 7(1), 43-60.

11. Samuel, P.; Mall, R.; and Kanth, P. (2007). Automatic test case generation from UML communication diagrams. *Information and software Technology,* 49(2), 158-171.
12. Chen, M.; Mishra, P.; and Kalita, D. (2010). Efficient test case generation for validation of UML activity diagrams. *Design Automation for embedded systems,* 14(2), 105-130.
13. Kumar, R.; Surjeet, S.; and Gopal, G. (2013). Automatic test case generation using genetic algorithm, *International Journal of Scientific and Engineering Research,* 4(6), 1135- 1141.
14. Sangeetha, S.; Ritu, S.; and Chayanika, S. (2011). Applying genetic algorithm for prioritization of test case scenarios derived from UML diagram. *International Journal of Computing Science*, 8(3), 433-444.
15. Aman, K.; and Rajeev, A. (2012). Applications of UML in real-time embedded system. *International Journal of Software Engineering and Applications*, 3(2), 59-70.
16. Gilberg, R.F.; and Forouzan, B.A. (2005). *Data structures: A pseudocode approach with C.* TATA McGraw Hill Edition, India.
17. Swain, R.; Panthi, V.; Behera, P.K.; and Mahapatra, D.P. (2012). Test case generation based on state machine diagram. *International Journal of Computer Information System,* 4(2), 99-124.