Test Case Generation For Concurrent Object-Oriented Systems Using Combinational Uml Models

Swagatika Dalai
School of Computer Engineering
KIIT University
Bhubaneswar

Arup Abhinna Acharya
School of Computer Engineering
KIIT University
Bhubaneswar

Durga Prasad Mohapatra
Department of Computer Science Engineering
National Institute of Technology
Rourkela

Abstract—**Software testing is an important phase of software development to ensure the quality and reliability of the software. Due to some limitations of code based testing method, the researcher has been taken a new method to work upon UML model based testing. It is found that different UML model is having different coverage and capable of detecting different kinds of faults. Here we have taken combinational UML models to have better coverage and fault detection capability. Testing concurrent system is difficult task because due to concurrent interaction among the threads and the system results in test case explosion. In this paper we have presented an approach of generating test cases for concurrent systems using combinational UML models i.e. sequence diagram and activity diagram .Then a Sequence-Activity Graph (SAG) is constructed from these two diagrams. Then that graph is traversed to generate test cases which are able to minimize test case explosion.**

Keywords-Software Maintenance; Regression Testing; Test case Prioritization.

I. INTRODUCTION

Software testing plays a vital role in System Development Life cycle. It is an investigation which is to be conducted to provide information about the quality of the product. According to IEEE testing is “the process of exercising or evaluating a system or system components by manual or automated means to verify that it satisfies specified requirements”. In other words testing is a process of gathering information by making observation and comparing them to expectations. The facts like the primary objective and the risks of software implementation are provided by software testing. Testing is not only meant for finding the bug in the code, but also it checks whether the program is behaving according to the given specifications and testing strategies [1]. If the software does not perform the required and expected result then a software failure is occurred. Therefore maximum of software development effort is being spent on testing. Generally the Testing process consists of three things: i) test case generation ii) test case execution iii) test case evaluation. The test case generation process plays a vital role among the three cases. The test case consists of three things i.e. input to the system, state of the system and the expected output from the system. If the test case detects maximum number of faults with minimum number of test cases then it is said to be having good coverage. During the software development process software testing can be implemented at any time. But the testing is implemented after all the requirements are defined and the coding process is over. But Code based testing having certain disadvantages which are as follows:

- Code based testing is not capable of extracting the behavioural aspects of the system.
- Code based software testing is not suitable for component based software development, because the source code may not be available to the developer.

An alternative approach is to generate test cases from the models which represent the software. It has the specific feature that the testing techniques can be applied throughout the development process depending on the requirement specification and design models. Another advantage of model based testing is that the generated test data is independent of any particular implementation of design. The model based testing reduces the testing time and effort.

Therefore now the researchers have used the analysis and design models like Unified Modelling Language (UML) for test case generation. UML models are very popular because UML is a solution of standardization and utilization of design methodologies. Another advantage of UML models is that it provide different diagram for representing different view of system models.

Concurrent computing is a property of systems in which several computations are executing simultaneously and also interacting with each other. Testing a concurrent system is a very difficult task because this type of system can reveal different responses depending upon different concurrency condition.

A concurrent system may be implemented via processes and/or threads. Due to concurrency a major problem arises known as explosion of test case because of the possibility for arbitrary interference of concurrent threads. These threads are executing relatively independently. However, since they are acting towards some goal, they must need to communicate and coordinate. So, issues arise while testing the concurrent system. In case of object oriented system objects interact with each other to accomplish task. While objects are interacting with each other there may be arbitrary interference of concurrent threads. Each thread may have more than one activity which may be dependent on each other; that may be inter thread dependency or intra thread dependency. Due to these problems there issues like Communication deadlock and synchronization may arise.

In this paper, we propose an approach for generating test cases using combinational UML diagram. In our approach we have taken the combination of sequence and activity diagram which are then traversed to generate the optimized test suite.

The rest of the paper is organized as follows: Section II describe the related work. The Basic concepts are described in section III. Analysis of our proposed methodology is discussed in Section IV. Section V contains the Conclusion and Future work.

II. Related Work

Sarma et al. [2] proposed a method for generating Test cases from UML Sequence Diagram (SD). In this technique the author first derive different operation scenarios from the Sequence Diagram. Here operation scenario means the set of messages that are flows between different objects. Based on this Scenario an intermediate format is constructed known as Sequence Diagram Graph (SDG). The state of the object changes when a message flows from one object to other. Here the author represents each state as a node and also assign an edge between the nodes. Here the graph has two ends i.e. one for true condition and other for false condition. Then the graph is traversed using graph traversal algorithms i.e. BFS and DFS. By applying All Sequence Message Path Criterion the author has find out all the possible message paths from the starting node to the end node. After traversal some test case are generated which are able to detect the operational faults as well as interaction faults.

Kim et al. [3] proposed a method to generate test cases from Activity Diagram (AD). The author first convert the Activity Diagram into an intermediate format known as called I/O explicit Activity Diagram (IOAD). In this diagram the author has suppressed the non-external input and output. He has only represented the external input and output as the internal activities are less important than the external activities to avoid test case explosion. Then an directed graph is being constructed using the basic path coverage criterion. After that the graph is traversed using DFS algorithm. Finally a set of basic paths are derived to generate the test cases.

Samuel et al. [4] proposed a method to generate the Test Sequences from UML 2.0 Sequence Diagram. In this approach the author first find out the different types of relationship like indirect message dependency, direct message dependency, simple indirect message dependency, simple direct message dependency that exists between the messages. Depending on the relationship different message sequences are generated and a graph is constructed known as Sequence Dependency Graph (SDG).Here each node in the SDG represent a message or a set of messages. Here the author associate node with the message number. Finally the SDG is traversed to generate the test cases.

The author Khandai et al. [5] proposed a method to generate test cases from UML 2.0 Sequence Diagram. To represent the complex scenarios she has used the Combined Fragment (CF). By applying some mapping rule In this approach first the Sequence Diagram (SD) is transformed into an intermediate form called Concurrent Composite Graph (CCG) to represent different scenario and their flow. Then the CCG is traversed using Message Sequence Paths Criteria (MSPC) to generate the test cases. The test cases are useful for detecting scenario, interaction as well as operational faults.

Sarma et al. [6] proposed a method for generating test cases from combination of UML models i.e. Sequence diagram (SD) and Use case Diagram (UD). In this approach first of all SD is converted into Sequence Diagram Graph (SDG). Then the UD is converted into Use case Diagram Graph (UDG.)In the UDG each actor is represented as node in the graph and assigns some edges between the nodes. Then the SDG and the UDG are combined to form a graph called as System Testing Graph (STG). Then STG is being traversed to generate the test cases.

The author Sun et al. [7] proposed a transformation-based approach to generate scenario oriented Test cases from UML Activity Diagram. The approach consists of three basic steps: First the UML Activity Diagram is transformed into an intermediate representation known as Binary Extended AND OR Tree (BET) via a set of transformation rule. The transformation rule is applied on different types of nodes like Fork node, Join node, Branch node and Merge node. Then the author applied an algorithm on the intermediate format to generate the Extended AND OR Tree. After that the tree is traversed using DFS algorithm to generate a set of test scenario. Finally from the test scenario a set of test cases are derived. The proposed this method for testing the concurrent system.

Kundu et al.[8] proposed an approach to generate test cases from UML Activity Diagram(AD). Then the AD is converted into an intermediate format known as Activity Graph (AG) by applying some transformation rules. The AG is traversed using BFS and DFS algo. BFS algorithm is used traverse all the concurrent activities and the rest are traversed by using DFS. Here the author uses the Activity Path Coverage Criterion to generate different Activity Paths.

Fan et al. [9] proposes a method for generating test cases from sub activity diagram to compound activity diagram in a hierarchical manner. They introduce the idea of this method by taking the thought of functional decomposition, bottom-up integration testing strategy and round-robin strategy.

Khandai et al. [10] proposed a method to generate test cases from combinational UML models such as Sequence Diagram (SD) and Activity Diagram (AD). In her approach AD is converted into an intermediate format known as Activity Graph (AG). After that test sequences are generated from AG by applying Activity Path Coverage Criteria. Then SD is converted into Sequence Graph (SG) and the test sequences are generated by applying All Message Path Coverage Criterion. For having better coverage and high fault detection capability the author constructed a Activity Sequence Graph (ASG) which has the combine features of AG and SG. Finally the ASG is traversed to generate the test cases.

III. Basic Concepts

UML Models

a) Sequence Diagram

The Sequence Diagram is a type of interaction diagram that is used for dynamic modelling which focuses on identifying the behaviour within the system. It represents object interaction arranged in a time sequence. These diagrams are used to represent or model the flow of messages, events and actions between the objects or components of a system. Sequence diagrams are typically used to describe the object-oriented system. It describes the objects and classes involved in the scenario and also the sequence of messages exchanged between the objects to carry out the functionality of the scenario.

b) Activity Diagram

An UML Activity Diagram is suitable for representing concurrent interaction among multiple threads. An Activity Diagram describes how multiple objects collaborate to do a specific set of operation. The basic elements of Activity Diagram are activity and transition. Activity is used as a state for doing something and the transition is represented as a directed line which connects different activities. A transition can be message flow, object flow or control flow. Activity Diagram is used for both conditional and parallel behaviour. Conditional behaviour can be denoted as a branch and a merge, and parallel behaviour is denoted by a fork and a join.

c) Concurrent System

In a Concurrent System different programs or threads are represented as collections of interacting computational processes that may be executed in parallel. The execution of threads begins from fork node and ends at join node. The main limitations in designing concurrent program to ensure the correct sequencing of the interactions or communications between different computational processes and coordinate the access to shared resources.

d) Activity path coverage criteria

This coverage criterion is used for both loop testing and concurrency among activities of activity diagrams. Here in this coverage criterion precedence relationship is maintained. An activity path is a path that allows a loop maximum two times and also maintains precedence relationship between different concurrent and non-concurrent activities.

IV. Proposed Methodology

We have proposed a methodology whose model diagram is shown in Fig. 1. The model has been proposed for generating test cases for concurrent system. In our approach we have presented an approach to generate test cases for concurrent system using combinational UML models. Here we have used the combination of Sequence diagram and Activity diagram. In a concurrent system several computations are processed simultaneously and also potentially interacting with each other.

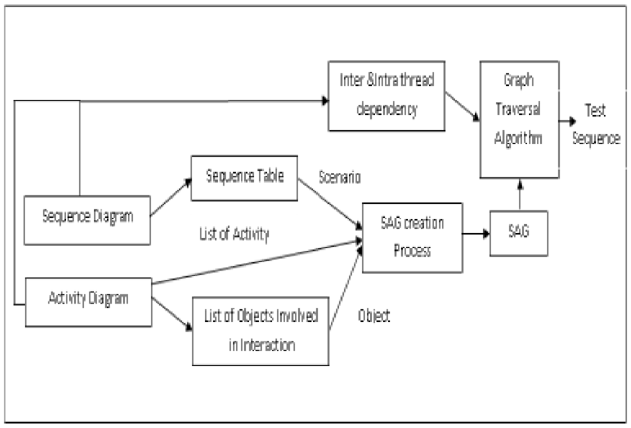


Figure 1. A Frame work of our proposed methodology

In our approach we have taken a sequence diagram and an activity diagram. From these two diagrams we have constructed a graph named as Sequence-Activity graph (SAG) by using an algorithm named as sequence-activity graph. Then the SAG is traversed using graph traversal algorithm i.e. BFS and DFS to generate test cases.

The resultant shows that the generated test cases are being capable of addressing the issue like test case explosion as we are dealing with concurrent system.

Concurrent computing is a form of computational process in which processes are designed as a collection of interacting computational processes that are executed in parallel. Here we have taken the sequence diagram because by using sequence diagram we can provide a dynamic view of the system in a graphical manner to display how the messages are passed between the objects at run time in order to perform tasks. Here we are dealing with concurrent system. Software testing is especially very much difficult when a system contains concurrently executing objects.

For this we have taken the activity diagram as the activity diagram is very much suitable for concurrent system as the activity diagram is capable of showing the parallel execution of different activities in a concurrent system.

Testing a concurrent system is a difficult task due to the arbitrary interference of different threads. Each thread may have more than one activity. Activities present in same thread maintain partial order relationship which shows dependency called as Intra thread dependency. Also activities of different thread may be dependent on each other called as Inter thread dependency.

Due to these types of dependency some critical situation arises during message passing which leads to a communication deadlock and also it causes test case explosion. So our objective is to minimize the test case explosion while generating the test cases.

Our approach consists of the following steps:

1) Construction of a Sequence Diagram (SD) and an Activity Diagram (AD).
2) Maintaining a Sequence Table (ST) with different schema as Source object(SC), Destination object(DO), Message ID(MI) and Message Content(MC) by taking the information from SD.
3) Then construction of Sequence-Activity Graph (SAG) by combining the features from SD and AD.
4) Finally traversal of SAG to generate test cases.

Example of Sequence diagram and Activity diagram

In this section we have presented the dummy examples of sequence and activity diagrams. Here we have used the Activity Diagram and Sequence diagram due to the following reasons.

Activity diagram is suitable for representing the concurrent activities as there is no state explosion of objects a in state chart diagram. Activity diagram shows the sequence of activity flows and also it represents parallel activities taken place in fork and join node.

Sequence Diagram (SD), also known as Interaction diagram represent the sequence of messages passed between the objects to specify some task without leading to test case explosion.

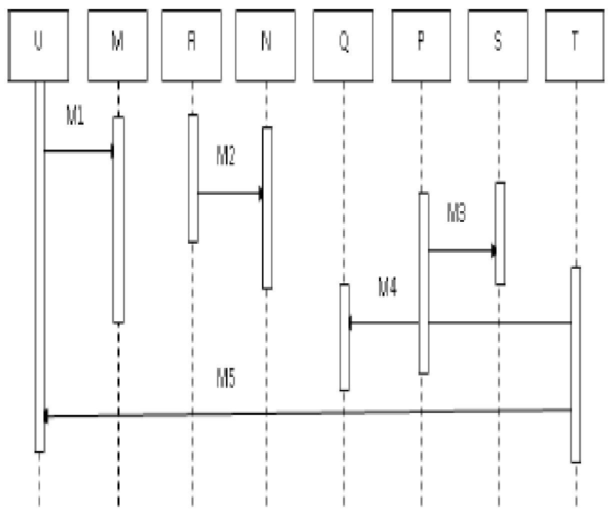
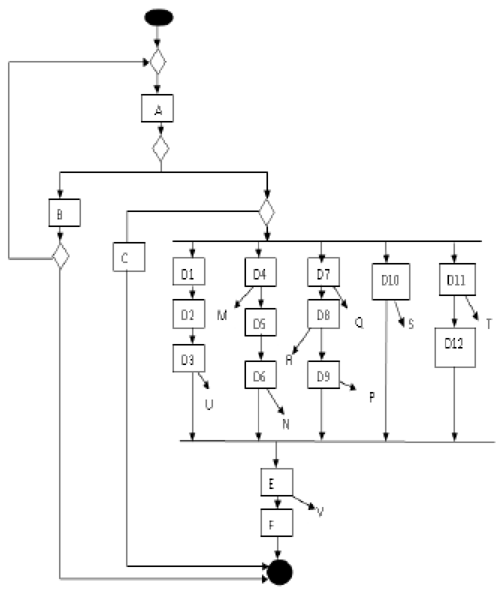


Figure 2. Sequence Diagram

U, M, R, N, Q, P, S, T are different objects of Sequence Diagram shown in Fig 2 . M1, M2 , M3,M4 and M5 are different messages that are flows between the objects.

The activity diagram represents the sequence of activity taken place. In this Activity diagram the Inter and Intra thread dependency are present. In above diagram different objects are like U,M,N,P,T,R,S,V are created after or before completion of different activities.



Technique for constructing Sequence Table (ST)

We have constructed a Sequence Table(ST) by taking the data from the Sequence Diagram as described in the section1.1.The table is constructed with the schema as Source object(SC), Destination object(DO), Message ID(MI) and Message Content(MC).

TABLE I. SEQUENCE TABLE

SO	DO	MI	MC
U	M	M1	aaa
R	N	M2	bbb
P	S	M3	ccc
T	Q	M4	ddd
T	U	M5	eee

Construction of SAG (Sequence-Activity Graph)

In this approach we propose a technique for generating SAG by combining both the features of Sequence diagram and Activity diagram. Whenever a transition is there we then take the two nodes in SAG and assign an edge between them. The nodes shown in the graph may be the Object node or Activity node. More no of threads are present in the Activity diagram in concurrent system. The activities present in different threads may be dependent on each other which are called as Inter thread dependency. The objects are created after activities are being taken place and communicating with each other by passing message among them. In the graph Inter thread dependency is represented by dashed arrows.

Node Representation of SAG

To represent a node in the graph two types of lists are required i.e. node list and edge list.

Node List

Status Bit	Node ID	Address of next node	Adjacent Node

Figure 3. Node Structure

Here the status field shows whether the node is a object node or activity node. 0 means object node and 1 means the node is an activity node. Node ID represents the unique ID of the node and the third field represents the address of the next node. and address of next edge shows the next adjacent node.

Edge List

Dest	Dependency Bit	Link

Figure 4. Edge Structure

Here the Dest field describes the destination link of the node and the Dependency Bit field describes the edge connecting to the nodes are Intra thread or Inter thread. I is denoted as Inter thread and O is denoted as intra thread dependency. Link field denotes the address of the next adjacent node of the Edge structure.

Traversal of SAG

Here we are using the graph traversal algorithm to traverse the graph and generate the test sequences by using the activity path coverage criteria.

```
Algorithm: SAG Traversal Algorithm
   Input: Sequence-Activity Graph
   Output: Set of test Sequence
    1.  Start.
    2.  Traverse the SAG using DFS.
    3.  While(Ncurr !=Nend)      //Ncurr=Current Node
        Nend=End Node
    4.  If(Ncurr= =Nf)           //Nf=Fork Node
        Then traverse the sub tree rooted at node Nf using
        BFS.

    5.  If Node List[Status bit= =1]
        Then traverse the edge list.
        Else
        Ncurr=Ncurr->next
    6.  If Edge List[Dependency bit= =0]
   Then there is inter thread dependency and traverse that
activity.
    7. Go to step4.
    8. Else Ncurr=Ncurr->next
     9. End If
    10. End while
    11. Exit.
```

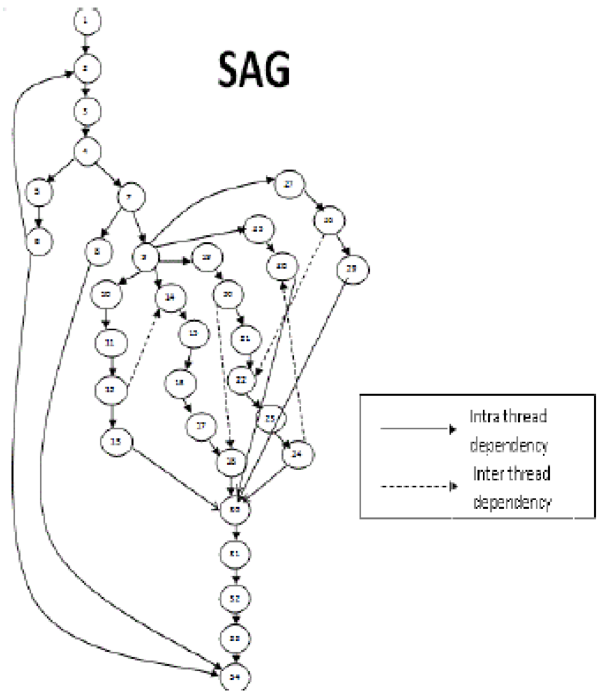


Figure 5. Sequence-Activity Graph (SAG)

After the construction of SAG we have traversed the SAG using Graph traversal algorithm. While traversing the nodes whenever a fork node is encountered, we will apply BFS (Breadth first Search) algorithm and for the rest nodes we have applied DFS (Depth First Search) algorithm in order to generate test cases. In this approach we have used the Activity path coverage criteria to generate test cases. After traversing the graph some test sequences are generated which are given below.

T1=1-2-3-4-5-6-2-3-4-5-6-34
T2=1-2-3-4-7-8-34
T3=1-2-3-4-7-9-10-11-14-12-13-15-16-17-18-19-20-21-22-23-26-24-25-27-28-29-30-31-32-33-34

V. CONCLUSION

In this paper we have proposed an approach for generating test cases using combinational UML diagram. In our approach we have taken the combination of sequence and activity diagram to construct a graph known as Sequence-Activity Graph(SAG).The partial ordering relationship that exists due to inter thread and intra thread communication is also included in the approach. The SAG is then traversed to generate the optimized test suite which minimizes the test case explosion.

VI. FUTURE WORK

The scalability of the approach is yet to be tested. A suitable optimization technique like Genetic Algorithm or Particle Swarm Optimization can be used to further reduce the volume of test data to be generated.

REFERENCES

[1] Rajib Mall, Fundamentals of Software Engineering, Prentice-Hall of India Comm, New Delhi, 2007.

[2] M. Sharma, D. Kundu, and R. Mal, Automatic test case generation from uml sequence diagram, In 15th International Conference on Advance Computing and Communication, IEEE, pages 60-65, 2007.

[3] H. Kim, S. Kang, J. Baik, and I. Ko, Test cases generation from UML activity diagram

[4] P. Samuel and A. T. Joseph, Test sequence generation from uml sequence diagrams, In Ninth ACIS International Conference on Software Engineering, Artificial intelligence, Networking, and parallel/distributed computing, IEEE, pages 879 - 887, 2008.

[5] M. Khandai, A. A. Acharya, D. P. Mohapatro, A Novel Approach of Test Case Generation for Concurrent Systems Using UML Sequence Diagram, 3rd International Conference on Electronics Computer Technology, ICECT, 2011, pages 157-161.

[6] M. Sharma and R. Mal, Automatic test case generation from uml models, In 10th International conference on Information Technology, IEEE, pages 196-201, 2007.

[7] C. Sun,A transformation-based approach to generating scenario-oriented test cases from uml activity diagram for concurrent applications, In Annual IEEE International Computer Software and Applications Conference, IEEE, pages 160- 167, 2008.

[8] D. Kundu and D.Samanta, A novel approach to generate test cases from UML activity diagrams, volume 8 no 3, pages 65 - 83, May-June 2009.

[9] X. Fan, J. Shu, L. Liu, and Q. Liango,Test case generation from uml subactivity and activity diagram ,In Second International Symposium on Electronic Commerce and Security, IEEE, pages 244 - 248, 2009.

[10] M. Khandai, A. A. Acharya, D. P. Mohapatro, Test Case Generation for Concurrent System using UML Combinational Diagram, International Journal of Computer Science and Information Technologies, IJCSIT,2011, Vol.2, Issue.

AUTHOR PROFILE

Swagatika Dalai is a faculty in Kalinga Polytechnique, KIIT University, Bhubaneswar, Odisha, INDIA. Her research areas include Object-Oriented System Testing. Now she is a M.tech student in KIIT University. She has a teaching experience of 5 years.

Arup Abhinna Acharya is an Assistant Professor and research scholar in the School of Computer Engineering, KIIT University, Bhubaneswar, Odisha, INDIA. He received his Masters degree from KIIT University Bhubaneswar. His research areas include Object Oriented Software Testing, Software Cost Estimation, and Data mining. Many publications are there to his credit in many International and National level journal and proceedings. He is having eight years of teaching experience. He is a member of ISTE.

Durga Prasad Mohapatra received his Masters degree from National Institute of Technology, Rourkela, India. He has received his Ph.D. from Indian Institute of Technology, Kharagpur, India. He is currently working as an Associate Professor at National Institute of Technology, Rourkela. His special fields of interest include Software Engineering, Discrete Mathematical Structure, Program Slicing and Distributed Computing. Many publications are there to his credit in many International and National level journal and proceedings.