IJCSE International Journal of Computer Science and Engineering Open Access
Research Paper Volume-2, Issue-3 E-ISSN: 2347-2693

Generation of Testcases from UML Sequence Diagram and Detecting Deadlocks using Loop Detection Algorithm

Amitashree Mallick[1*], Namita Panda[2] and Arup Abhinna Acharya[3]

[1*]*School of Computer Engineering, KIIT University, India, amita29dec@gmail.com*
[2] *School of Computer Engineering, KIIT University, India,npanda@kiit.ac.in*
[3]*School of Computer Engineering, KIIT University, India, aacharya@kiit.ac.in*

www.ijcseonline.org

Received: 11 March 2014 Revised: 14 March 2014 Accepted: 26 March 2014 Published: 31 March 2014

Abstract— In an environment where processes those execute concurrently, speeding up their computation is important. Deadlock is a major issue that occurs during concurrent execution. In this paper, we present an approach to generate testcases from UML sequence diagram for detecting deadlocks during the design phase. This will reduce the effort and cost involved to fix deadlocks at a later stage. Our work begins with design of sequence diagram for the system, then converting it to intermediate graph where deadlock points are marked and then traverse to get testcases. The testcases thus generated are suitable for detecting deadlocks.

Keywords— *Software testing, Test cases,Sequence diagram, Concurrency, Deadlock*

I. Introduction

Testing is the most important part of quality assurance in software development life cycle. As the complexity and size of software products grow, the time and effort required to do effective testing increases. Studies indicate that more than half of software development cost is devoted to testing[1]. If the tests are in process before implementation, costs of software development will be reduced.

Unified Modelling Language(UML) is the most dominant standard language used in modelling the requirement and considered an important source of information for testcase design. A sequence diagram is one of the UML diagrams that is helpful in modelling object interaction.
Concurrency is a property of system 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 testcase explosion. Also synchronization and deadlock create problems when concurrently running objects want to interact with each other.

Generation of testcases can be done manually by an experienced test engineer or automatically by a testcase generation tool if the required application knowledge is sufficiently formalized. If the application is large and complex then manual testing will require more time compared to done automatically. Another problem in testcase generation is we cannot test the testcases until they are actually run.
This paper uses UML sequence diagram as design specifications. Sequence Diagram can be useful because it results in less number of test cases. The sequence diagram is also useful in detecting scenario as well as interaction faults. Our approach first transforms a sequence diagram to an intermediate graph and then identifies the possible deadlock points in it. Then the graph is traversed to generate testcases using loop detection algorithm useful to detect deadlock. By detecting deadlocks in the design phase of software development process the cost of software development can be reduced.

The rest of the paper is organised as follows: Section II describes the background, Section III focuses on related work, Section IV summarizes the proposed approach, Section V highlights testcase generation, Section VI gives a comparison with related work and finally section VII presents the conclusion.

II. BACKGROUND

In this section, we describe the basic concepts that are used in this paper on model based testing, UML 2.0 Sequence diagrams, deadlocks, sequence diagram graph and testcases that are essential in analysing the present method.

A. Model Based Testing

Model based testing consists of the requirement specifications as input to generate test cases[4].Three main reasons for using model based testing are:
(1) Traditional software testing techniques consider only static view of code, which is not sufficient for testing dynamic behaviour of object-oriented softwares ,
(2) Use of code to test an object-oriented software is a complex and tedious task
(3)Model-based test case generation can be planned at an early stage of the software development life cycle, allowing software developer to carry out coding and testing in parallel[3][7].

Corresponding Author: *Amitashree Mallick, amita29dec@gmail.com*

B. Sequence Diagram

A sequence diagram(SD) is very useful in representing sequence of actions that occur in the system over time. It easily represents the invocation of methods from each object. Sequence diagram consists of objects and messages. Objects are represented using rectangular boxes and messages using arrows[7]. In this paper we will use SD to model our system.

C. Deadlock

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. A Deadlock can be of two types:
Communication Deadlock: This occurs when process A is trying to send a message to process B, process B trying to send a message to process C which is turn tries to send a message to A.

Resource deadlock: This occurs when processes try to have exclusive access to resources e.g. devices, files, locks, servers, etc..Resource deadlock can be represented by resource allocation graph. If the graph contains a cycle then it may have a deadlock.

III. RELATED WORK

Many testing works are based on UML models, proposing concepts for generation of testcases and scenario coverage based system testing. Those works are suitable for system testing and detecting various faults but do not take into consideration issues due to concurrency in communication and deadlocks. Sarma et al[5] presented an efficient method for testcase generation from UML 2.0 sequence diagrams by converting to an intermediate graph. The testcases generated from the graph are able to detect interaction and scenario faults. Samuel et al[6] enlists the relationships that can exist among messages and efficiently generate test sequences from UML 2.0 sequence diagrams. This too converts the sequence diagram to an intermediate graph whose nodes are incorporated with message sequences. The graph is then traversed to get the test sequences. Cartaxo et al[7] uses a systematic procedure of functional testcase generation for feature testing. The procedure is based on model based testing techniques with testcases generated from UML SDs translated into label transition systems. Feature testing is important as costs to find bugs during the feature interaction phase are higher since feature interaction testing phase involves more than one feature. Nagarani et al[2] provides a very good approach on automation of software testing process using Coded UI(User Interface) tool. This method will provide better utilization of resources and save time. Wenming et al[10] proposes an integration of various testing tools for system operation. The system also integrates test data so that all independent test data are integrated into a whole structure, easy to manage and maintain. Khandai et al[8] proposes an approach for generating. testcases which consists of transforming the sequence diagram into a concurrent composite graph. This approach is effective in controlling the testcase explosion problem and detecting interaction, scenario, and composite faults in concurrent systems. Patnaik et al[9] represents a deadlock situation in a concurrent process using SD and also lays testcases for it. It is relevant because detecting deadlocks in the design phase will help reduce the effort which would have been otherwise spent in the implementation phase to detect deadlock and debug it.

IV. PROPOSED APPROACH

The block diagram of the proposed approach is shown in Fig 1.It starts by modelling a sequence diagram for the application and follows through the steps to generate testcases. Our aim is to generate testcases that can detect deadlocks. So we have used UML 2.0 sequence diagram that will help in representing concurrency and result in lesser testcases. ’par’ combined fragment has been used in SD to represent concurrent activity.

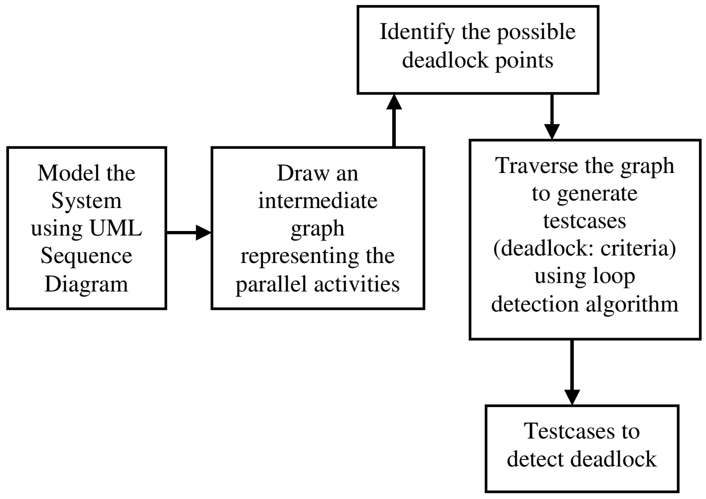
Model the System using UML Sequence Diagram
Draw an intermediate graph representing the parallel activities
Identify the possible deadlock points
Traverse the graph to generate testcases (deadlock: criteria) using loop detection algorithm
Testcases to detect deadlock

Fig 1: Block Diagram of proposed Approach

The approach consists of five steps. After representing appropriate SD, an equivalent wait for graph is represented which helps to identify the deadlock points. The wait for graph is then traversed using a Tarjan’s cycle detection algorithm[11] to get the cycles. Then the approach proceeds towards generation of testcases. For this we find the operation scenarios from SD and draw the sequence diagram graph(SDG).SDG has a start state and two end states to mark deadlock and non-deadlock state. The steps are explained in later with example.

A. Step 1: Sequence Diagram

Sequence diagram used in our work is shown in Fig 2.Sequence Diagram consists of objects and messages. Objects are represented using object names in rectangular boxes which interact with other objects. All messages are represented as simple messages as a arrow marked from sender to receiver with individual message names. a, b, c, d and e are objects.m1( ),m2( ),m3( ),m4( ),m5( ),m6( ),m7( ) and m8( ) are messages.

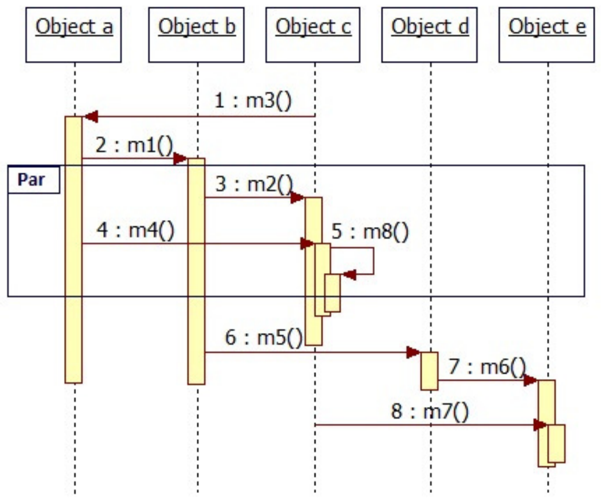

Fig 2: Sequence Diagram

Concurrent activity is shown using 'par' combined fragment in the sequence diagram(SD).The messages inside the fragment m2( ),m4( ),m8( )try to execute concurrently.

B. Step2: Wait for Graph

Deadlock is identified with the help of a wait for graph. It consists of processes as vertices and messages as edges. Wait for graph for the system is shown in Fig 3. Wait for graph is a simple directed graph that shows possible deadlocks between processes as cycles.

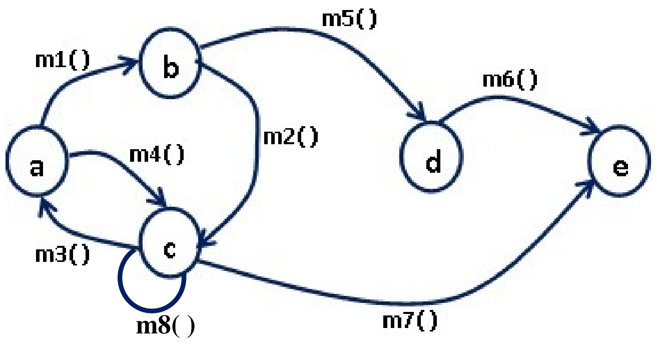

Fig. 2: Wait for Graph

In the graph an edge from 'a' to 'b' indicates 'a' is waiting for an event to be completed by process 'b'. As we have discussed earlier a system contains a deadlock if it consists of a cycle. In Fig 3 there are three cycles, a →b →c →a, a →c →a and c →c. In Fig 2 message m2( ),m4( ) and m8( ) are executing concurrently. Processes a, b and c each wait to perform events by other processes(its next node) which are busy waiting for events to occur from another process(next node) in cycle. These go into a hold and wait condition and give a circular loop causing deadlock.

C. Step 3: Identifying all deadlock points

We use Tarjan's algorithm[14] to find all deadlock paths in the system from the wait for graph. This traversal algorithm is based on depth first search. The vertices are indexed as they are traversed by DFS procedure. While returning from the recursion of DFS, every vertex V gets assigned a vertex L as a representative. L is a vertex with the least index that can be reach from V. Nodes with the same representative assigned are located in the same strongly connected component.

D. Step 4: Operation Scenarios

The next step is to derive the operation scenarios from SD. The operation scenario is shown in Fig 4.

<scn1	**<scn2**	**<scn3**	**<scn4**	**<scn5**
State X	State X	State X	State X	State X
S1: (m3,c,a)	S1: (m3,c,a)	S5: (m8,c,c)	S1: (m3,c,a)	S1:(m3,c,a)
S2: (m1,a,b)	S4: (m4,a,c)	State Y>	S2: (m1,a,b)	State Z>
S3: (m2,b,c)	State Y>		S6: (m5,b,d)	
State Y>			S7: (m6,d,e)	
			State Z>	

Fig. 4: Operation Scenarios

There are 5 operation scenarios here. scn1...scn5 denotes scenario number, state X is the starting state and state Y, state Z are end states. s_i (i=1...7) is the state number. m_j (j=1...8) is the message between two objects. The procedure for operation scenarios has been adopted from Sarma et al[5].

E. Step5: Converting SD to SDG

To draw an SDG we have to note all the operation scenarios and events that occur in it from start state to end state. The SDG is shown in Fig 5.There is one start state(state X) and two end states(state Y and Z).All scenarios that do not form a loop end at state Y and others at Z. In other words end state Z indicates a safe state and end state Y indicates a deadlock state.

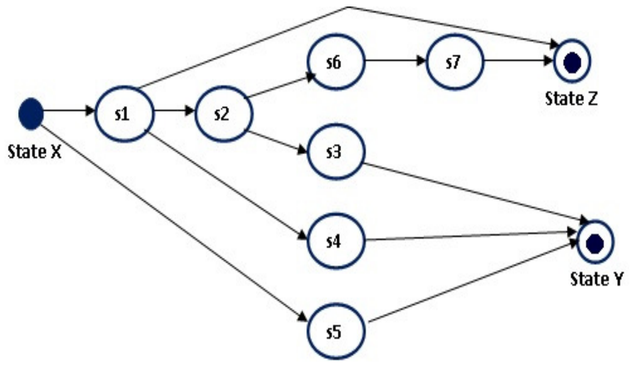

Fig 5: Sequence Diagram Graph

F. Step6: Generation of testcase

To generate testcases all paths from start node to end node are enumerated and each path is considered a testcase. Our test targets to detect concurrency fault called deadlock.

Traversal of SDG to generate all testcases is done using testcase generation algorithm given below.

Algorithm: Testcase Generation

Input: Sequence Diagram Graph of Sequence Diagram

Output: Deadlock paths and test suite(T)

Steps:

```
1. TP[ ]=identify All Paths(STG) //Enumerate all Basic Paths
TP=TP[1],TP[2],...,TP[N] from s node to end node in STG
2. For each path TP[i]∈TP do
3. current node(CN)=S(start node)
4. preCondition=Precondition of scenario //Precondition is
stored in N
5. $TC_i$= Φ//The testcase of $scenario_i$ is initially empty.
6. while(CN≠FN)do //CN is current node, FN is the final
node, Path is i
7. $event_{CN}$=<m,a,b,c> //Event corresponding to current
node .m( ) is invoked with a set of arguments a,b,c
8. if CN=Guard then
9. Select testcase
//TC={preCondition, Inputs, Outputs, postCondition}
10.Add TC to the testset $TS_i=TS_i \cup TS$
11.End if
12.If C≠Guard then
13.If $C_{value}=\{ C_1,C_2,...,C_p \}$
TC={preCondition, Inputs, Outputs, postCondition}
The set of value of classes on the path $P_i$
14.Select testcase
TC={preCondition, Inputs, Outputs, postCondition}
15.Add TC to test set $TC_i$, that is, $TC_i=TC_i \cup$ TC
16.End if
17.CN=next node //Move to the next node $N_k$ on the path $P_i$
18.$TC_i=TC_i \cup TC$
19. End while
20.Determine the final output and $postCondition_i$ for the $scn_i$
stored in final node.
21.TC={preCondition, Inputs, Outputs, PostCondition}
22.Add the testcase TC to the test set TS that is
TS←TS ∪ TC
23.End for
24.Return(TS)
25.Stop
```

Each path in the SDG is traversed to get a testcase. Traversal is done from 'start' node to 'end' node. There are two states Y and Z. State Y indicates path reaching it may lead to deadlock and state Z indicates non-deadlock paths. Step 21 gives all the testcases corresponding to the scenario as a whole. Testcases using the above algorithm is mentioned in section V.

Following are the testcases that were finally derived using the proposed approach.

1.Testcase="Dummy Model"

2.Precondition:C is the initial state where execution states.

3.Testcases:Scenario 1

Input: message 3

Output: message 2

Postcondition: Deadlock state

4.Testcase:Scenario 2

Input: message 3

Output:message 4

Postcondition: Deadlock State

5.Testcase:Scenario 3

Input: message 8

Output: message 8

Postcondition: Deadlock State

4.Testcase:Scenario 4

Input: message 3

Output: message 6

Postcondition: Non-deadlocked Path

5.Testcase: Scenario 5

Input: message 7

Output: message 7

Postcondition: Non-deadlocked state

V. COMPARISION WITH RELATED WORK

From the related work we come to realise that there are many efficient methods for testcase generation from UML SDs[5][7][8][9] and automation testing[2][10].The method proposed in [5] is an efficient method for testcase generation but doesn’t take into consideration concurrency faults although it detects interaction and scenario faults. In [6] it presents a very good procedure for test sequence generation which is important for correct sequential execution of test cases. Nagarani et al[5] also provides us a good concept on how their developed tool is helpful in automating complex software applications. Wenming et al[2] proposes another concept on automation testing and how automation tools can be integrated to easily manage and maintain the testing process. In [8][9] they discuss on solving concurrency and synchronization issues and generating testcases to detect them early in the design phase.

VI. CONCLUSION

In this paper, we proposed a method for generating testcases from UML sequence diagram. It mainly consisted of 3 steps model UML sequence diagram, representing the sequence diagram graph and traversing it to generate testcases that will be able to detect deadlocks. The used example was a generalised model that shows the test can exercise all paths and has the ability to detect deadlocks. In future the paper intends to use tools for automatically generating those testcases and implement it in a real-time scenario. More work needs to be done to combine other UML diagrams in our method.

VII. BIBLIOGRAPHY

[1] R.V Binder, "Testing Object-Oriented Systems Models, Patterns, and Tools, Object Technology Series". Addision Wesley, Reading, Massachusetts, October 1999

[2] P. Nagarani I, R. Venkata Ramana Chary, "A Tool Based Approach For Automation Of GUI Applications",ICCCNT’12 26th-28th July 2012, Coimbatore, India

[3] N. S. Dsouza, A. Pasala, A. Rickett and O. Estrada,"A code based approach to generate functional test scenarios for testing of re-hosted application", Short Papers of the 22nd IFIP ICTSS, Alexandre Petrenko, Adenilso Simao, Jose Carlos Maldonado (eds.), Nov. 08-10, 2010, Natal, Brazil

[4] Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall, "Test Case Generation Based on Use case and Sequence Diagram", Journal of Object Technology, vol. 8,No. 3,May-June 2009,PP. 65-83

[5] Monalisa Sarma, Debasish Kundu, Rajib Mall, "Automatic Test Case Generation from UML Sequence Diagrams".15th International Conference on Advanced Computing and Combinations,2007 IEEE

[6] Philip Samuel, Anju Teresa Joseph, "Test Sequence Generation from UML Sequence Diagrams",2008 IEEE

[7] Emanuela G.Cartaxo, Francisco G.O.Neto and Patriticia D.L.Machado,Test Case Generation by means of UML Sequence Diagrams and Labelled Transition Systems, 2007 IEEE

[8] Monalisha Khandai,Arup Abhinna Acharya,Durga Prasad Mohapatra, A Novel Approach of Test Case Generation for Concurrent Systems Using UML Sequence Diagram978-1-4244-8679-3,2011 IEEE

[9] Debashree Patnaik,Arup Abhinna Acharya, Durga Prasad Mohapatra, Generation Of Test Cases Using UML Sequence Diagram In A
System With Communication Deadlock, Journal of Computer Science and Information Technologies, Vol. 2 (3) , 2011, 1187-1190

[10] Guo Wenming, Fu Xiangling, Feng Jianmei, "A Data-driven Software Testing Tools Integration System", School of Software Engineering, Beijing University of Post and Telecommunication,Beijing,P.R.China,2009

[11] Srinivasan Desican, Gopalaswamy Ramesh, Software Testing principles and practices,3rd Edition, Pearson Publication

[12] Rajib Mall ,Software Testing principles and practices,3rd Edition, Pearson Publication

[13] Srinivasan Desican, Gopalaswamy Ramesh, Fundamentals of Software Engineering,3rd Edition, PHI Learning Private Limited

[14] Tarjan, R. E. ,Depth-first search and linear graph algorithms, SIAM Journal on Computing 1 (2): 146160,1972 doi:10.1137/0201010

AUTHORS PROFILE

Amitashree Mallick is a student of Mtech in Computer Science and Information Security, KIIT University, Bhubaneswar, Odisha, INDIA. Her research area is Automated Testing. She can be reached at amita29dec@gmail.com

Namita Panda is an Assistant Professor in the School of Computer Engineering, KIIT University, Bhubaneswar, Odisha, INDIA. She received her Master’s degree from KIIT University Bhubaneswar. Her research areas include Object Oriented Software Testing, Parallel Processing and Computer Architecture. She has published papers in national and international level proceedings. She is having ten years of teaching experience. She is a member of ISTE. She can be reached at npandafcs@kiit.ac.in.

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 eleven years of teaching experience. He is a member of ISTE. He can be reached at aacharyafcs@kiit.ac.in.