Journal of Advanced Research in Cloud Computing, Virtualization and Web Applications
Volume 1, Issue 2 - 2018, Pg. No. 19-22
Peer Reviewed Journal

Research Article

State Oriented Software System Testing for Object Oriented Applications using UML Diagrams

Deepak Nagar[1], Jitendra Kumar[2]

[1,2]Assistant Professor, IIMT College of Management, Uttar Pradesh, India.

Abstract

Object-oriented software depends upon the successful integration of classes and object. When the classes are integrated to each other, there could be chance to arise several faults. The method in the area of research is SCOTEM (State Collaboration Test Model), based on UML collaboration and State chart diagrams. This is a state-based approach which generates the various test path based on the coverage criteria Selected and hence includes all the objects states in collaboration.

In this paper the analysis of SCOTEM model is done for a case study using a prototype tool developed in C language. Various Mutants are analyzed by using this prototype tool. The results show that the related technique effectively detects all the seeded faults when complying with the most demanding adequacy criterion and still achieves reasonably good results for less expensive adequacy criteria.

Keywords: SCOTEM, M(Message), Ob(Object)

Introduction

Five distinct level of object oriented testing.[2]

- A method,
- Message quiescence,
- Event quiescence,
- Thread testing,
- Thread Interaction Testing

The communicative nature of objects[2]

As shown in fig 1 the interaction among the objects in object-oriented program may be between the methods within the same object or between the methods of two different objects show in integration testing the emphasis is on object-oriented interaction[1] rather than in side the objects.

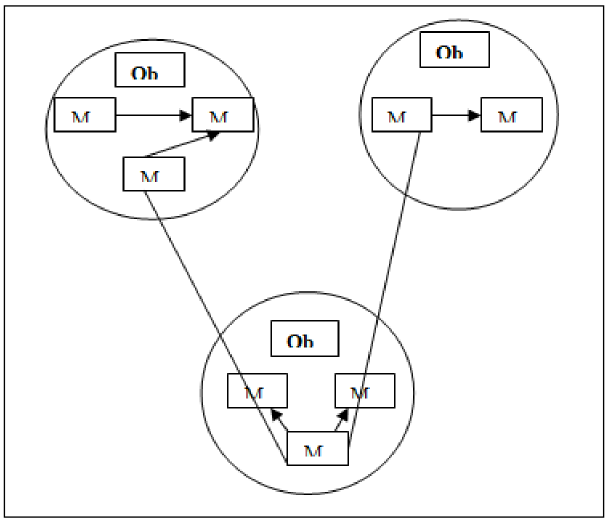

Figure 1.Message communication[2]

State Collaboration Test Model (SCOTEM)[1]

The SCOTEM is an intermediate test model which is used for automatically generate test specifications for class integration testing. The Various phases in SCOTEM is given below.

Corresponding Author: Deepak Nagar, IIMT group of colleges, India.
E-mail Id: deepg.ngr@gmail.com
Orcid Id: https://orcid.org/0000-0003-3642-983X
How to cite this article: Nagar D, Kumar J. State Oriented Software System Testing for Object Oriented Applications using UML Diagrams. *J Adv Res Cloud Comp Virtu Web Appl* 2018; 1(2): 19-22.

- SCOTEM Generation: An intermediate test model, called SCOTEM (State Collaboration Test Model) is constructed from a UML collaboration diagram and the corresponding state charts.
- Test Paths Generation: Test paths are generated from the SCOTEM based on multiple possible alternative coverage criteria.
- Test Execution: All specified test paths are executed by using manually-generated test data and an execution log is created, which records object states before and after execution of each message in a test path [3]. The object states are determined using state invariant assertions.
- Result Evaluation: The object states in the execution log are compared with the expected object states in the test paths generated from SCOTEM. This means that these test paths also contain oracle information in the form of expected states of the objects. If any state of any object after execution of a test path is not in the required resultant state, then the corresponding test case is considered to have failed.

Where the short words means the following

CD=Collaboration Diagram
SC=State Chart
SG=Scotem Generator
TPG=Test Path Generator
CC=Coverage Criteria
TP=Test path
SI=State Invariant
TD=Test Data
TE=Test Executor
P/f Res=Pass/Fail Result

The SCOTEM is a particular graph structure: A vertex corresponds to an instance of a class (in a particular state) participating in the collaboration.

A Modal Class can receive a message in more than one state and exhibit distinct behavior for the same message in different states.[7] To capture this characteristic, for modal classes, the SCOTEM contains multiple vertices, where each vertex corresponds to an instance of the class in a distinct abstract state (corresponding to states defined in state charts). At the same time, a non-modal class only requires a single vertex in the SCOTEM graph.

SCOTEM test model's edge can be segregated in two types: message and transition edges. A message edge reflects a call action between two objects, and a transition edge reflects a state-transition of an objection receiving a message. Each message edge may also contain a condition or iteration. Every message can be a reason of a state transition to occur.

A transition edge connects two vertices of the same class. State charts may have multiple transitions to distinct states for the same operation. Hence, there can be several transition edges (representing a conditional state transition) for the same message edge in SCOTEM. Each of these transitions is usually controlled by mutually exclusive scenario (to prevent non-determinism). The internal representation of a vertex holds the class name and state of the instance it corresponds to. Message edges are modeled in the SCOTEM by attributes of a message including message sequence number, associated operation, receiver object, and the sender object.[4] The transition edges are modeled by the attributes of a transition which includes sequence number, associated operation, accepting state and sending state.

Mutants for Integration Testing[6]

The following mutant operators are being used while performing integration testing

- Replace Return Statement
- Remove Function Call
- Condition Missing
- Loop Error Set
- Alter Condition Operator
- Guard Condition Violated
- Missing Called Function
- Target State as Source State
- Wrong Calling State
- Conflicting State Operator

Case Study

In order to validate system testing approach using the prototype tool developed[6], we consider here a simple example of Student Evaluation System for case study through out the work.

The implementation of Student Evaluation System that we consider in our example evaluates the grade of the students. The application presents generates questions, one after the other, to the students. Students are given lot of time to solve each problem. A detailed performance log is maintained for each student. The classes under the test are shown in Table. 1.

Table 1.Classes under the Test

Sr. No.	Class	Method	Instance variable	Moda lity	Lines of code
1	Login User	1	2	Non Model	29
2	Display Question	2	2	Non Model	36
3	Log	1	2	Modal	27
4	Result	1	2	Modal	28
5	Grade Obtained	1	3	Modal	32

The collaboration diagram of the system is shown below in the diagram.

1: Start (): Void
1.1: Question (int 1): Void
1.1.1: Set Log (Int): Void
1.1.2: Set Result (int): Void
1.1.2.1:Set Grade (int count)

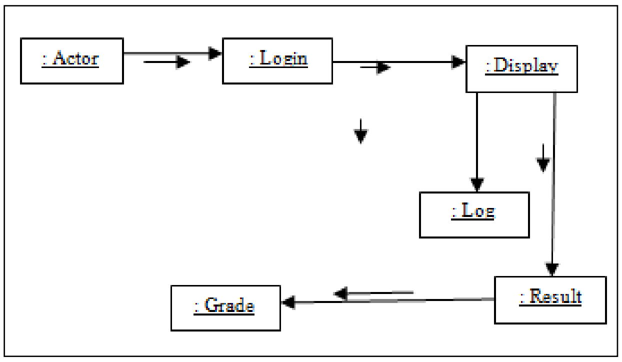

Figure 2.Collaboration Diagram of System

The following are flattened state diagram of different classes. All the UML diagrams are made in tool Poseidon4.5.

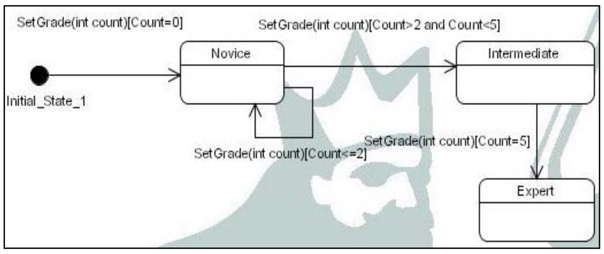
SetGrade(int count)[Count=0]
SetGrade(int count)[Count>2 and Count<5]
Novice
Intermediate
Initial_State_1
SetGrade(int count)[Count=5]
SetGrade(int count)[Count<=2]
Expert

Figure 3.StateChart Diagram of Class Grade

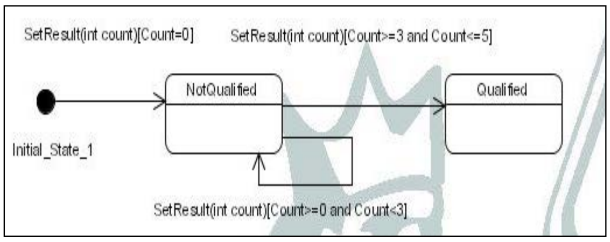
SetResult(int count)[Count=0]
SetResult(int count)[Count>=3 and Count<=5]
NotQualified
Qualified
Initial_State_1
SetResult(int count)[Count>=0 and Count<3]

Figure 4.StateChart Diagram of Class Log

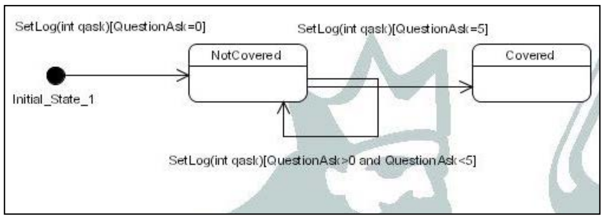
SetLog(int qask)[QuestionAsk=0]
SetLog(int qask)[QuestionAsk=5]
NotCovered
Covered
Initial_State_1
SetLog(int qask)[QuestionAsk>0 and QuestionAsk<5]

Figure 5.StateChart Diagram of Class Result

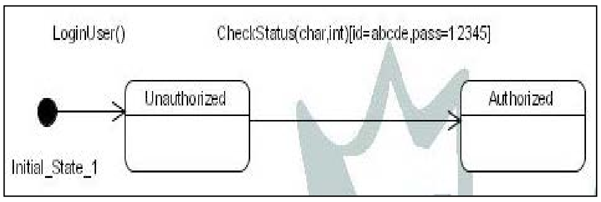

Figure 6.StateChart Diagram of Class Login

Algorithm

The following is algorithm is used for creating graphical structure of SCOTEM model

Files used are

SCOTT.txt: is used to give input to this algo.

FILE pointer variables are

in :used to point the file SCOTT.txt

Procedure and subprocedures used are

:procedure void xstrcpy(char *d,char *s)
:procedure int search(char *seq_no)
:procedure void storeobj(struct node **p,char *obj)
:procedure void storestate(struct node **p,char *st)
:procedure void storetransition(struct node **p,char *guard,char *source,char *target)
:procedure struct node * storemsg(char *seq_no)
:procedure void createpath(struct node *p)
:procedure void store(struct node **p,char *seq)
:procedure void print(struct node *p)
:procedure void deletestate(struct node *h,char *state_name)
:procedure int getkey()
:procedure void push(struct stack **top,struct node *save)
:procedure struct node* pop(struct stack **top)
:procedure void print_stack()

Variables used are

name : is data member of structure state used to store name of state
*next : is data member of structure state used to point next state
cond : is used for condition of transition
source : is used for transition source
target : is used for transition target
feasible: is data member of structure transition to check transition from source to target is feasible or not
*next : is data member of structure transition pointer to next transition
*current:is variable of structure node pointing to first node

Algorithm for creating SCOTEM graph void scotem(struct node *ptr)

```
*ptr=NULL
store(&ptr,”1”)
head=ptr
deletestate(head,”Hidden”)
createpath(head)
d=DETECT
initgraph(&d,&g,”c:\\tc\\bgi”)
void scotem(struct node*)
l=strlen(“1”)*8+20
```

```
.maxx=getmaxx()
maxy=getmaxy()
        midx=maxx/2
        midy=maxy/2
        Initialize the X and Y coordinates of all states and
iteratives
        *current=head
        instruction()
        getkey()
        scotem(current)
        while(1)
do select case(getkey())
                case 80:exit(0)
                case 72:if(top==NULL)
                                then break
                                current=pop(&top)
                        scotem(current)
                case 75:if(current->left!=NULL)
then push(&top,current)
                                current=current->left
                                scotem(current)
                case 77:if(current->right!=NULL)
                            then push(&top,current)
                                current=current->right
                                    scotem(current)
closegraph()
```

Algorithm for xstrcpy

```
void xstrcpy(char *d,char *s)
        while (*s!='\n' && *s!='\0' )
                do *d=*s
                        s++
                        d++
        *d='\0'
```

Algorithm for search

```
int search(char *seq_no)
char *seq=(char*)malloc(sizeof(char)*strlen(seq_no)+3);
strcpy(seq,"'");
strcat(seq,seq_no);
strcat(seq,":");
if((in=fopen("SCOTT.txt","r"))==NULL)
then perror("Unable to open file: ");
exit(0);   while(fgets(string,250,in)!=NULL)
```

Conclusion

This study represents here used to validate the already available approaches by the prototype tool we developed. We also implemented here the several path coverage criteria to validate the approach.Once we redirect about the future work in this model then we must discuss the test result generated by this prototype tool for system testing. We can also use this technique to test any SOA kind of application.

References

1. Shaukat Ali, Lionel C.Briand, Rehman MJ, et al. A State based approach to integration testing based on UML models. *Journal Information and Software Technology* 2007; 49(11&12): 1087-1106.
2. Zhao R, Lin L. An UML State chart Diagram-Based MM-Path Generation Approach for Object-Oriented Integration Testing. World Academy of Science, Engineering and Technology, *International Journal of Computer and Systems Engineering* 2008; 2(10): 3470-5.
3. labiche Y, P.thevenod-Fosse, Waeselynck H et al.Testing Levels for Object-oriented Software. 2000 in ICSE, 2300 1-58113-206-9/00/06.
4. Bruegge B, Dutoit AH. Object-Oriented Software Engineering: Using UML, Patterns and Java, Prentice Hall, Second Edition, 2003.
5. Baldini A, Benso A, Prinetto P. System-level Functional Testing from UML Specifications in End-of-production Industrial Environments. *International Journal of Software Tools Technology and Transfer* 2004; 7(4): 326-340.
6. Briand L, Labiche Y, Wang Y. An Investigation of Graph-Based Class Integration Test Order Strategies, *IEEE Transactions on Software Engineering* 2003; 29(6): 594 - 607.
7. Briand LC, Di Penta M, Labiche Y. Assessing and Improving State-Based Class Testing: A Series of Experiments. *IEEE Transactions on Software Engineering* 2004; 30(11): 770-79.

Date of Submission: 2018-12-13
Date of Acceptance: 2018-12-20