IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 3, Ver. 1 (May – Jun. 2015), PP 17-24
www.iosrjournals.org

Model Based Software Timing Analysis Using Sequence Diagram for Commercial Applications

Hrishikesh Mukherjee[1], Shrimann Upadhyay[2] ,Arup Abhinaa Achariya[3]
[1,2,3](CSE, School of Computer Engineering/KIIT University, India)

Abstract: *The verification of running time of a program is necessary in designing a system with real life constrain. Verification defines lower and upper bounds which reflects control flow that depends on data as well as instruction times execution. In previous days, the bounds were very wide due to a lack of efficient control flow analysis and architectural modeling techniques. But in present days, significant progress have been occurred in both areas in a way such that execution cost of formal software analysis has become more practical. In previous days various work has been done on code based complexity analysis for embedded software. This type of analysis is called as worst case execution time analysis(WCET). In our research we are applying the concept of timing analysis for UML model based timing analysis, by which we can minimize the effort of calculating timing from code snippet and it will also help the developer to identify the timing requirement since requirement stage.*

Keywords: *Verification, Worst Case Effective Time, UML Models*

I. Introduction

Timing analysis is a method to estimate uninterrupted execution time of an application on given hardware. It is a crucial component for performance analysis[1] of an application. It is also a crucial subtask for design-space exploration, application scheduling and binding. Generally execution time of an Instruction depends on the architecture it runs. Execution time can be obtained from processors datasheet either from basic instructions or can be derived from composite instructions. Execution time may not be constant. It depends on interrupts, cache effects, pipeline stalls, branch prediction, and many other factors. So we need methods for timing analysis. The process to determine timing of an execution of a program is called as Worst Case Effective Time(WCET)[10]. It is an upper bound on the run time of an given application on a given hardware. It determine an upper bound on execution time for many possible inputs. We can computer WCET from two application specific model like sequential terminating process and input-output based model. There are many simulator by which we can compute WCET. It motivated a static analysis-based approach. There are three types of WCET observation like Estimated WCET, Observed WCET, Actual WCET. For analyzing WCET determine the control flow graph from code snippet and analyze it. It is a very effective process to determine the proper run time of a program in various architecture and environment.

II. Related Work

In [11], they showed an process analysis nature by applying running time interval. It is being done by discovering program segments using single flow paths and considering execution time context into account. This analysis basically experimented on embedded applications, where systems are dependent on process state and input data. These running time interval depends on program properties, execution paths, process states and execution architecture. In [13], timing analysis of UML sequence diagram deals with real time systems. In this approach, timing analysis is given with one scenario without any interaction constraints means without any loops or alternative flow of control. They mainly developed a time consistency checking algorithm, that takes into account of composition of UML sequence diagram, which describes various scenarios. In [12], they demonstrated the timing analysis by linear programming and made an algorithm using integer time verification technique. Practicality of this approach works for primary pillar of verification of real time applications.

III. Proposed Framework For Software Timing Analysis

In this paper, a framework on software timing analysis has been proposed on UML Model based approach. Here first requirements are analyzed and gathered in form of UML 2.4[2,5,7] diagrams and from UML Sequence diagrams, a UML graph[4] is formed and after applying few algorithms the maximum and minimum working time taken for the application is counted. The advantage of this approach is that from the requirement stages we are able to decide an **abstract working time of a software.** Firstly project requirements[6] are gathered after which a sequence diagram is constructed out of the elicited requirements. Then the particular sequence diagram is converted to the label transition graph. The graph is then reduced to path expression. After that minimum and maximum time determined an aspect of path lengths[9].

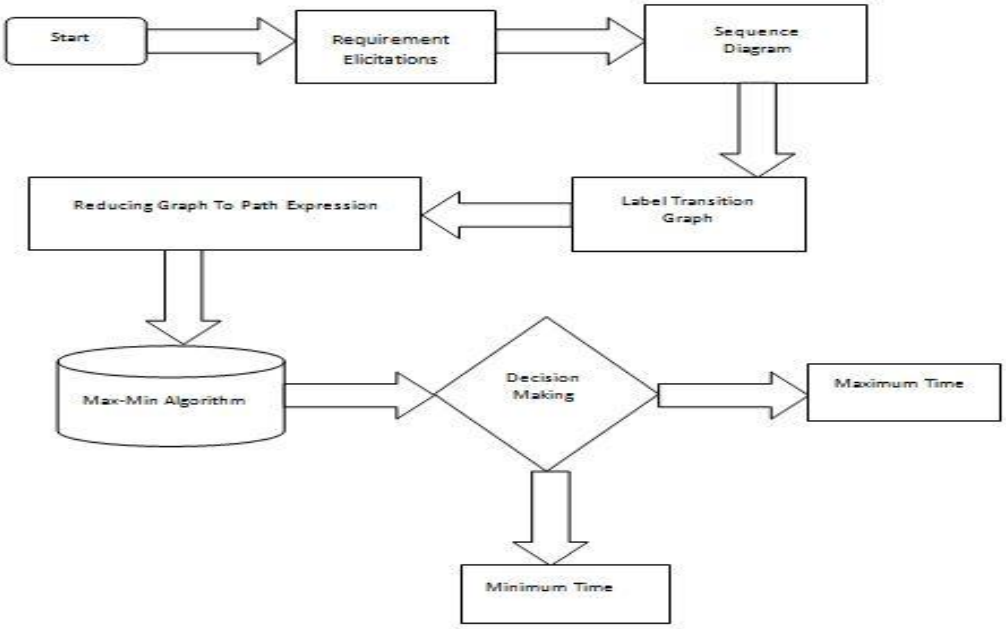


Fig1: Proposed framework for software timing analysis model

Here maximum path means maximum time taken for the working of the application.
Main activities for software timing analysis are noted step by step

- First we convert sequence diagram into a label transition graph(LTG)[8].The steps of the LTG are created in ascending order. The initial state is numbered '0' and it is associated with an input arrow in its left.
- For each messages in sequence diagram we apply the following procedure. If the flow of control from an user to system or from user to user, then generate a transition with label named 'steps' , where label is the message content. If the flow of control is from system to user then make a transition with label named 'expected _result'.
- For alternative flow of control, we generate a transition with label 'condition' and for each case of conditional looping, create a transition with label, which is the proper loop and other form the ended loop state to the first loop state with the same level.

3.1 Sequence diagram:

It is a type of interaction diagram which represents the interaction between objects through messages. In sequence diagram object is represented by rectangle. Dashed line represents its lifeline and the vertical bar represents the active part. Here objects communicate through the messages, represented by horizontal arrows drawn from message sender to message recipient. There are two types of messages. One is synchronous & other is asynchronous. A simple message generally has a normal flow of control form one object to other with a direction, is called a synchronous message. In asynchronous messages continue its working without the completion or the delivery of the messages. This messages are modeled as half of an open arrow next to message text that is being sent.
A sequence diagram contains the following elements:-Object, Messages, Lifeline, Combined Fragments
Object:- A sequence diagram consists of sequence of interactions between various objects and objects are represented by rectangle.
Messages:- The relation or interaction between objects are viewed by messages. A message is marked by a direct arrow.
Lifeline:- In a sequence diagram , the lifelines basically are controlled rectangles which are drawn as vertical hollow rectangles over the objects lifeline. The control rectangle notation is a simple notation that helps the reader to understand when an object is involved in a sequence of messages. In most cases, an object has a focus at a time: however, in event- driven applications with asynchronous functionality, this is not always the case.

2.2 Combined Fragments:

Combined fragment is an interaction fragment which defines an expression of interaction fragments. It is defined by an Interaction operator and corresponding interaction operands. Combined fragment may have interaction constraints also called GUARDS in UML 2.4. Interaction operator could be one of:

alt(alternative)
opt(option)
loop(iteration)
break(break)
par(parallel)
alt:- The interaction operator alt means that the combined fragment represents a choice of alternative behavior. In this case the chosen operand must have an explicit or implicit GUARD expression that evaluates to true at this point in the interaction. An implicit true GUARD is implied if the operand has no GUARD. An operand guarded by else means a GUARD that is the negation of the isolation of all other GUARDS. If none of the operands has a GUARD that evaluates to true, none of the operands are executed and remainder of enclosing interaction fragment is executed.
opt:- The interaction operator opt represents a choice of characteristics in optional way where either operand happens or not happens to be present. An option is semantically equivalent to an alternative combined fragment where there is one operand with non-empty content and the second operand is empty.
loop:-loop means we are repeating an action over a number of times. The interaction operator loop means it is a special type of combined fragment which basically represents a loop. It represents a recursive application of the seq operator where the loop operand is sequenced after the previous iterations. Loop operand may contain iteration bounds which may include a lower and upper number of iterations of the loop. Textual syntax of the loop is
loop-operand::=loop['('min-int[','max-int]')']
min-int ::=non-negative integer
max-int::=positive-integer|'*'
If the loop has no specified bounds, it means potentially infinite loop with zero as lower bound and infinite as upper bound.
If only min-int is specified it means that upper bound is equal to the lower bound, and loop will be executed exactly the specified number of times.
If max-int is specified, it should be greater than or equal to min-int. In this case the loop will iterate minimum the min-int number of times and at most the max-int number of times.
break:-break represents an exceptional scenario that is performed instead of the remainder of the enclosing of interaction fragment. A break operator with a guard is chosen when the guard is true. In this case the rest of the directly enclosing fragment is ignored. When the guard of the break operand is false, the break operand is ignored and the rest of the enclosing interaction continues. It is interesting to note that UML allows only one level interaction fragment. Double loop or loop with other combined fragments are never taken in case of break or it can be said that break is never used for double loop or loop with other combined fragments. Another interesting feature in UML2.3 is that if there is no guard then interaction fragment is unpredictable.
par:-par defines parallel execution of behaviours of the operands of the combined fragment. Parallel combined fragment has notational shorthand for the common situations where the order of events on one lifeline is insignificant. In a co region area of a lifeline restricted by horizontal square brackets all directly contained fragments are considered as separate operands parallel combined fragment.

2.3 Label Transition System:
Label Transition System provides a global, monolithic description of the set of all possible nature of the system; a path on the LTS can be measured as a test sequence. Therefore LTSs are highly testable models. In the following figure the elements of the LTS are shown below:-
a) Initial state represent an initial state of the system;
b)Label Transition that represents an action that occurs and change the state of system;
c)State that represents a state of a system;
An LTS is a four tuples $S=(Q,A,T,q_0)$ where

Q is a finite, non-empty set of states;
A is a finite, non-empty set of labels (denoting actions);
T, the transition relation, is a subset of Q x A x Q;
q_0 is the initial state.

In this paper, a framework on software timing analysis has been proposed on UML Model based approach. Here first requirements are analyzed and gathered in form of UML 2.4[2] diagrams and from UML Sequence diagrams, a UML[4] graph is formed and after applying few algorithms the maximum and minimum working time taken for the application is counted. The advantage of this approach is that from the requirement stages we are able to decide an **abstract working time of a software.**

Firstly project requirements[6] are gathered after which a sequence diagram is constructed out of the elicited requirements. Then the particular sequence diagram is converted to the label transition graph. The graph is then reduced to path expression. After the graph is reduced to path expression then, in the next stage the minimum and maximum path lengths are determined. Here minimum path length means minimum time taken for the working of the application and maximum path means maximum time taken for the working of the application.

2.4 Sequence Diagram Into Label Transition Graph:

1)Each steps of the LTG are created and numbered as an increasing order, according to the transition.
2)In LTG the initial state in numbered as '0' and an input arrow is used to mark the start node.
3)From each message in sequence diagram apply the following-
-From User to System or from User to User, generate a transition with the label 'steps' and a transition where label is expressed as proper message content.
-From System to User, create a transition with the label 'expected_ results' and a transition where label is the proper message content(i.e. if you found any recursion then label it as a 'expected_ results').
4)For alternative flow of control, create a transition with label 'condition' and the transition associated with it.
5)For each condition loop, a transition must be created where label is the proper loop and other from the ended loop state to the first loop state to the first loop state with the same label .

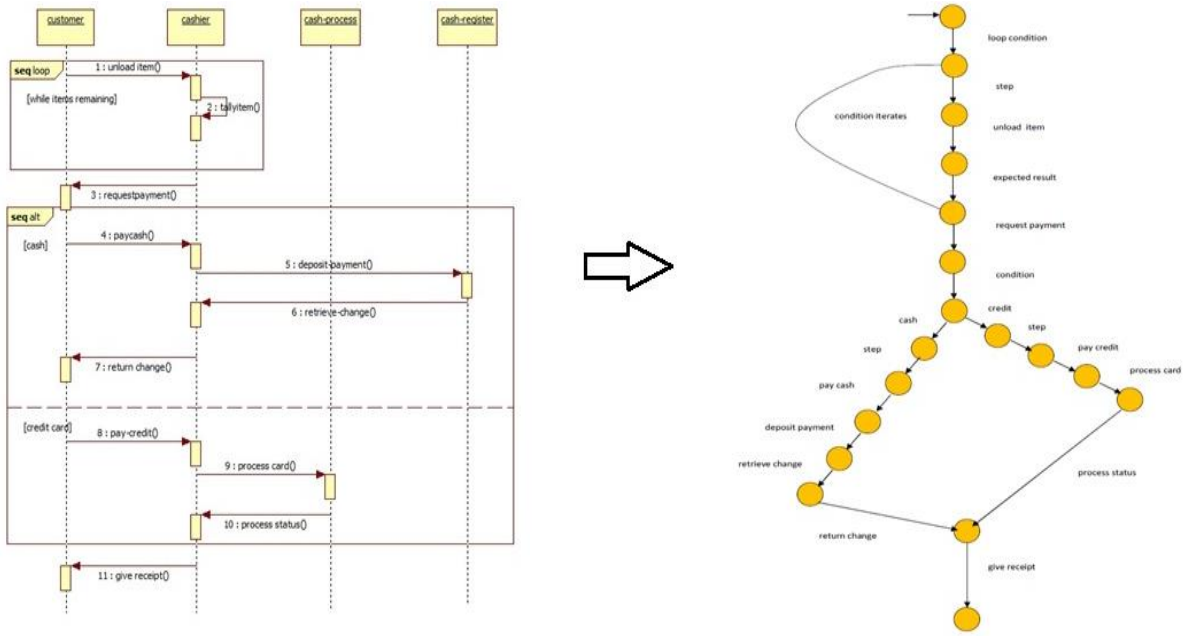
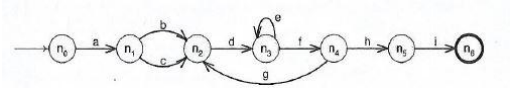


Figure 2: Sequence digram with label transition graph for purchasing process in a mall

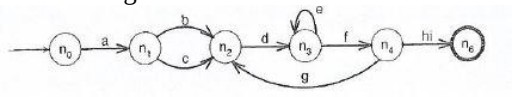
The conversion of a sequence diagram to a label transition graph is shown below:

2.5 Reduce Graphs To Path Expression:

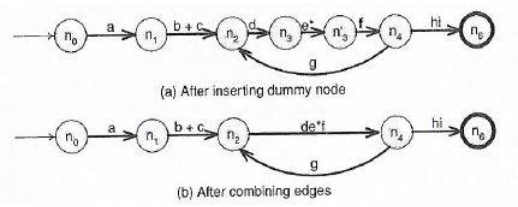
1)For sequentially executing edges, multiply the edge label.



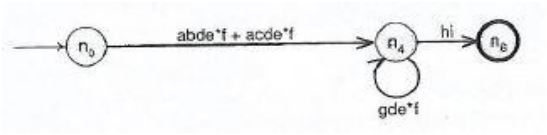
2)Combine all parallel edges and add the edges levels. More



3)If we found any self loop, then mark the node using exponent operator '*'.



4)Finally we get the following expression as shown in the figure below.



3.2 Proposed Algorithm:

Algorithm max_ time()
Input: Weight of edges of label transition graph
Output: Maximum time

{
Let two consecutive edges A and B and edge weights are W_A , W_B;
W_A and W_B are initialised as 1;
1. Begin
2. for each edges e that belongs to (A,B,.....,n)
3. if path expression is A+B
4. then substitution is max(W_A,W_B).
5. end of if
6. if path expression is AB
7. then substitution is W_A+W_B.
8. end of if
9. if path expression is A^n
10. then substitution is n*W_A.
11. End of if
12. End of for
13. end
}

Algorithm min_ time()
Input: Weight of edges of label transition graph
Output: Minimum time

{
Let two consecutive edges A and B and edge weights are W_A , W_B;
W_A and W_B are initialised as 1;
1. Begin
2. for each edges e that belongs to (A,B,.....,n)
3. if path expression is A+B,
4. then substitution is W_A+W_B.
5. End of if
6. if path expression is AB
7. then substitution is min(W_A,W_B)
8. End of if
9. if path expression is A^n,
10. then substitution is either 1 or W_A.
11. End of if
12. End of for
13. End
}

3.3 Case Study of Timing Analysis Using Sequence Diagram for Purchasing Process in a Mall

Determining the maximum time:

max{loop*condition*step*unload item*expected result*request payment*condition iterates*condition*((cash*step*pay cash*deposit payment*retrieve change*return change)+(credit*step*pay credit*process card*process status))*give receipt}

= loop + condition + step +unload item + expected result + request payment + condition iterates + condition + (max(cash*step*pay cash*deposit payment*retrieve change*return change) + max(credit*step*pay credit*process card*process status)) + give receipt

= 7 + (max (cash*step*pay cash*deposit payment*retrieve change*return change) + max (credit*step*pay credit*process card*process status)) + 1

= 8 + ((cash + step + pay cash + deposit payment + retrieve change + return change) + (credit + step + pay credit +process card + process status))

= 8 + (6 + 5) = 8 + max(6,5) = 8 + 6 = 14 time units

Determining the minimum time :

min{loop*condition*step*unload item*expected result*request payment*condition iterates*condition*((cash*step*pay cash*deposit payment*retrieve change*return change)+(credit*step*pay credit*process card*process status))*give receipt}

Now for advantage of our calculation , we are decomposing the expressions in some smaller manners.

min((cash*step*pay cash*deposit payment*retrieve change*return change)

= max(1,1,1,1,1,1) = 1

min(credit*step*pay credit*process card*process status)

= max(1,1,1,1,1) = 1.so, min(1+1) = (1 + 1) = 2 Now , finally the result is: max(1,1,1,1,1,1,1,2,1)= 2 time units

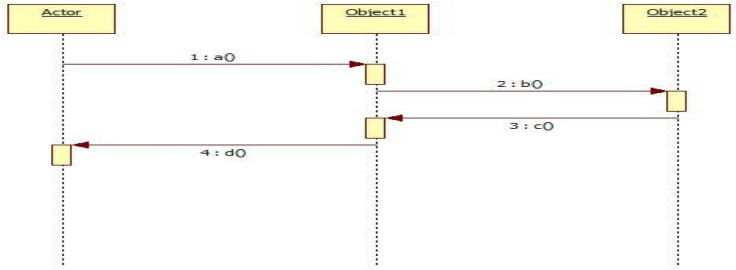


Figure 3: Sequence diagram with synchronous call & reply messages

IV. Experimental Evaluations

Table 1: Timing table with respect to $\sum IM$ and MaxT

Object	$\sum IM$	MaxT
3	4	5
4	6	7
5	8	9
6	10	11
7	12	13

We take an example of sequence diagram with synchronous call with reply messages(including only one actor). Suppose a sequence diagram has 3 objects('O'). Each object is associated with a synchronous call with reply messages. Total number of interaction messages are 4. If we follow the same procedure for 4 objects(including actor itself as an object), the total number of interaction messages are 6. Now if we take 5 or 6 or more objects likewise, then the total number of messages will be in increasing and for 5 messages it would be 8 and so on. Now from the above table we can observe that total number of interaction messages('IM') are in increasing manner with the number of objects. It is occurring in a particular pattern and we can formulate it.

$$\sum IM = 2*(O-1) \quad \text{.....(1)}$$

A graph is shown below to demonstrate the relationship between number of objects versus number of total interaction messages. The number of messages increase linearly with the number of objects are increasing.

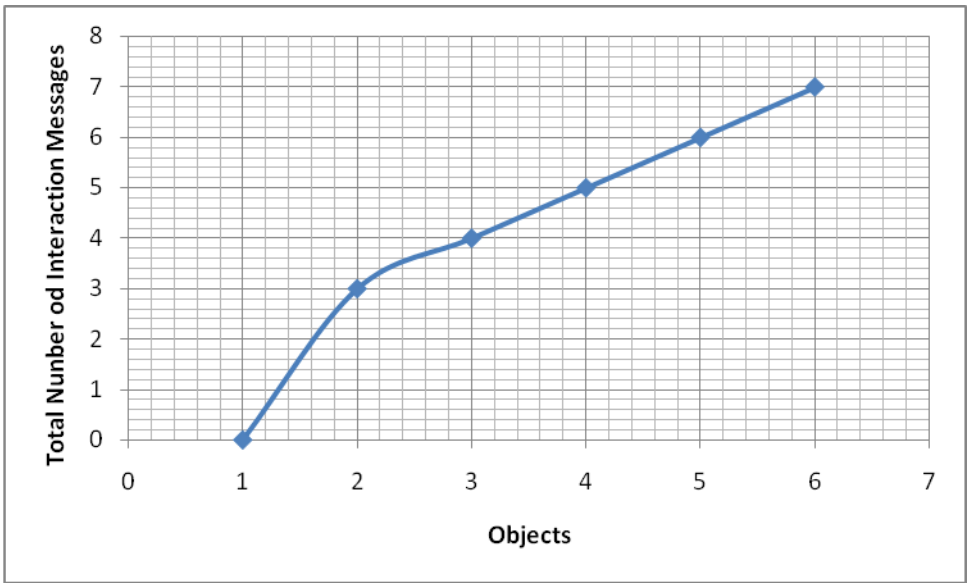


Figure 4: Relationship between object and $\sum IM$

Now, calculate the timing by the following method. For 3 objects total number of IM's are 4 and for starting node one 'step' needed, so MaxT will be (4+1) = 5 time units. So for 4 or more objects, it follows the same procedure. From the above table we can formulate timing with respect to the number of objects. Here in all cases MinT will be constant The required formula will be :

$$MaxT = (2 * O - 1).....(2)$$

Where, $MaxT$ = maximum time
$MinT$ = minimum time

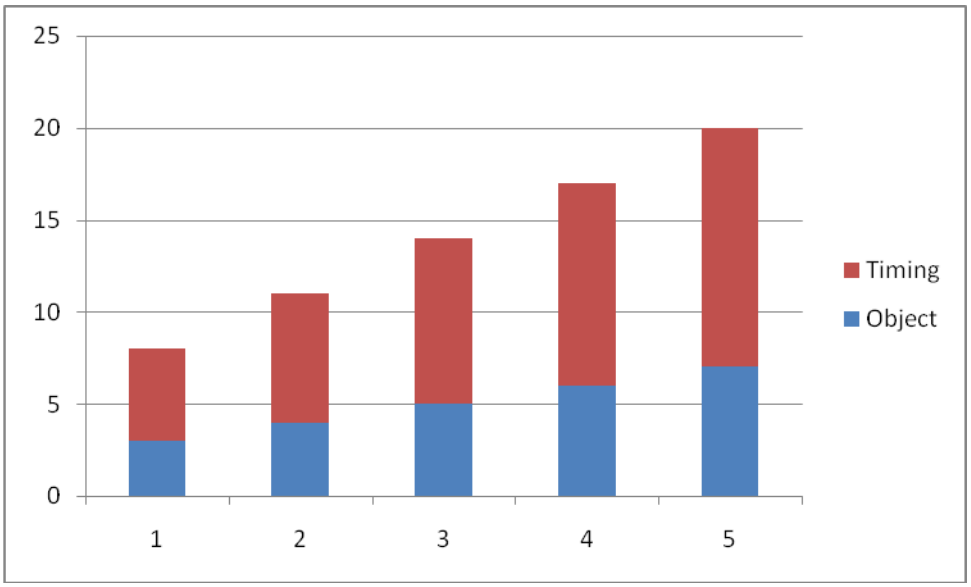


Figure 5: Relationship between object and timing

After analyzing equation (1) and (2), we extract a relationship between interaction messages and MaxT. It is described as a formula. A graphical representation in shown below:

$$\sum IM = MaxT - 1.....(3)$$

3.5 .Analysis of Timing with More Interaction Constraints and Messages:

In the above diagram for purchasing system, total number of objects = 4.So, total number of interaction messages $\sum$ IM = 2(4-1) = 6, MaxT = (2*4 - 1) = 7, MinT = 1.

The above calculations only occurs, when sequence diagram has synchronous call with reply messages associated with each object. But in practical cases, sequence diagram contain more or less interactions messages

and constraints. In our primary calculation total interaction messages are 6 and practical consideration, it is 14. So, there is 8 more time units. The above scenario we can represent by the following formula:

$$MaxT = (2 * O - 1) \pm K \text{.....(2)}$$

Where, K is a variable with any integer value range between $\{1 \leq k \leq 100\}$

$$MinT = 1 \pm K$$

Where, K is a variable with any integer value range between $\{0 \leq k \leq 1\}$

V. Conclusion

Quality of software cannot be ensured without an effective testing strategy which compromises timing analysis also. So, we plan to do more survey on timing analysis and to propose a model and develop an algorithm for UML Model based timing analysis. Finally we will try to implement timing analysis which will include project deployment timing. In the current approach we have only shown an numerical approach for our analysis method. In future we will try to implement a tool based upon our propose methodology. First we will try to generate a LTS graph from sequence diagram using XML schema based approach, then proper coding will be needed to implement the analysis. Then accuracy analysis of our procedure will come also in our topic.

References

[1]. Y .Li and S. Malik, Performance Analysis of Real-Time Embedded Software, Norwell, MA: Kluwer, 1999.
[2]. Object Management Group: UML 2.0 specifications, OMG adopted specification 2003, "http://www.omg.org.
[3]. A. V. Aho, R. Sethi, J. D. Ullman, Compilers: Principles, Techniques & tools, Reading, MA: Addison-Wesley, 1992.
[4]. Hartmann, M. Vieria , H. Foster and A. Ruder, A UML-based Approach to System Testing, Journal of Innovation System Software Engineering, Vol. 1, PP. 12-14, 2005.
[5]. D Jeya Mala, S Geetha, Object Oriented Analysis and Design Using UML, McGraw Hill Education.
[6]. Ian Sommerville, Requirements Engineering: A Good Practice Guide, Wiley, 2000
[7]. D. Pilone and N. Pitman, UML 2.0 in a nutshell, O'Reilley, NY, USA, 2005.
[8]. E. G. Cartaxo, F. G. O. Neto, P. L. Machado, "Test Case Generation By Means Of UML Sequence Diagram and Label Transition Systems", In Proceeding Of IEEE International Test Conference 2007.
[9]. Paul Ammann, Jeff Offutt, Introduction to Software Testing, Cambridge University Press, First Edition.
[10]. A. Banerjee, Worst Case Execution Time Analysis For Embedded Program, Invited Talk in National Conference On Recent Trends On Software Testing 2014.
[11]. F. Wolf, R. Ernst, W. Ye, "Path Clustering In Software Timing Analysis", IEEE Transaction On Very Large Scale Integration (VLSI) Systems, Vol. 9, No. 6, December 2001
[12]. Li Xuandong, Cui Meng, Pei Yu, Zhao Jianhua, Zheng Gualiang, "Timing Analysis of UML Activity Diagram", Springer-Verlag Berlin Heidelberg 2001
[13]. Li Xuandong, Johan Lilihus, "Timing Analysis of UML Sequence Diagram", Turku Center for Computer Science, TUCS Technical Report No 281, May 1999, ISBN 952-12, 0466-4, ISSN