A Survey on Test Case Generation from UML Model

Monalisha Khandai#1, Arup Abhinna Acharya#2, Durga Prasad Mohapatra*3

School of Computer Engineering, KIIT University
Bhubaneswar, India

**Department of Computer Science & Engineering*
National Institute of Technology
Rourkela, India

Abstract— **Testing is an important phase of software development, to maintain the quality control and reliability of the end products. Recent approach has been taken by the researcher to use UML models for test case generation. Various works has been done on test case generation for concurrent and nonconcurrent systems. In case of concurrent system group of activities are executed simultaneously where as in case of non-concurrent system the activities are executed sequentially. Some of the work has also been done for generating test cases from combinational UML models. In this paper we have gone through a survey on test case generation from UML models. Our works focus on finding the existing process of test case generation from UML model/s for concurrent as well as nonconcurrent systems. This paper will make help to the researcher interested in the field of test case generation from UML model to find out what work has been done in their interested field. We have gone through 15 articles which have been published in the time span of 2005-2010.**

Keywords— **Testing, TestCases, concurrent system, nonconcurrent system, UML Models.**

I. INTRODUCTION

Testing is an important phase of software to produce high reliable system and to maintain quality control. The reliability and quality of the end product depend to a large extend on testing. Therefore more than 50% of software development effort is being spent on testing. 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 word testing is the process of identifying the difference between the expected and actual results. If the software does not perform as required and expected then a software failure is said to be occurred. Testing effort consists of three things: i) test case generation or selection ii) test case execution iii) test case evaluation. Among the three, test cases generation problem is receiving highest attention. A test case is normally a triplet [I, S, O], where "I" is data input to the system "S" is the state of the system to which the data will input, and "O" is the expected output from the system. A test case is said to be having good code coverage if it uncovers/detect maximum number of faults with minimum number of test cases and having high fault detection capability. Combination of all the test case with which a given software product is to be tested is called test suite.

Depending on the testing method employed, Software testing can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. But since code based testing have certain disadvantages over model based testing. So model based testing being an alternative approach became popular allowing the testing technique to be applied along with the development phase.

Recent approach that has been taken by researchers is to use analysis design models like Unified Modelling Language (UML) for test case generation. UML models are very popular because when software engineering industry was in desperate need for standardization and utilization of design methodologies, UML came up as a solution. Other advantage of UML models is that it provide different diagram for representing different view of system models and it is easy to automate. Automated test case generation is advantageous when we have to generate the test cases for large system which is inherently complex. In such a case generating all the large number of test cases and carrying out the test cases is very time consuming and labour intensive. The automated test generating tool can be helpful in such a cases by saving the time and cost. There are different tool available such as QTP, Rational Rose of IBM for generating the test cases automatically. But the recent approach can generate the test cases semiautomatically.

Though many work has been done for sequential testing a few work has been done for concurrency testing. Testing concurrent systems is a very crucial task since such a system can exhibit different responses depending on the concurrency conditions. Due to concurrency there may be test explosion. Synchronization and deadlock create problems when concurrently running objects want to interact with each other. The UML Sequence Diagram, Activity Diagram and State Chart Diagram can be used for testing concurrency. However State Chart Diagram is useful for unit testing and results a large number of test cases where as the Sequence Diagram can be useful for integration testing resulting a less number of test cases. The Activity Diagram is useful for representing complex sequence of parallel and conditional activities.

In this paper literature survey is done on various methodologies available for generating test cases from UML models for concurrent as well as non concurrent systems. A summary on the work that has been done on this field, their advantage and disadvantages has been presented. Performing a literature search helps to define an unsolved problem. Literature survey helps to have adequate knowledge of what has been produced in the area of interest which is the most crucial aspect for a researcher.

The rest of the paper is organized as follows: Section II represents the various methodologies available for generating test cases from UML model or combinational UML models. Section III discusses the advantages and disadvantages of the techniques and finally section IV discusses the conclusions and future work.

II. TEST CASE GENERATION FROM UML DIAGRAMS

In this section we will discuss various techniques available for test case generation. First we will consider the technique available for generating test cases from single UML model followed by the technique available for generating test cases from combinational UML models.

A) From Single UML Diagram

In the following subsection we will first discuss the research techniques available for generating test cases from single UML models for nonconcurrent testing (sequential testing), followed by the techniques available for concurrent testing. There are various UML models available such as UML Sequence Diagram, Activity Diagram, Statechart Diagram and Communication Diagram.

1) For non-concurrent systems

Samuel et al. [1] present a method to generate test cases based on UML Communication Diagram. Their approach consists of the following steps:

a) Convert the Communication Diagram into Communication Tree.
b) Traverse the tree in post order manner to select the Conditional Predicates.
c) Transform the Conditional Predicate.
d) Generate the test data.

This technique first converts the Communication Diagram into a Communication Tree. The Communication Diagram consists of two things first one is *node* which is represented by rectangles and the other is the *edge*. The rectangle depicts the object and the edge or the link between the object depict the message passed between the objects. Since the communication diagram doesn't have the time as a separate dimension so the messages have ordered using sequence number in edge. For example if the first message passed from the object A to B then it is numbered as **0**. Then if the next message sequence is passed from B to C then it is numbered as **1.** Suppose after that there are two messages from object C then one of them will be ordered as **1.1** and the other will be ordered as **1.2**. While constructing the communication tree the edge plays a major role. Since edge represents the message sequence along with the message. So considering the message sequences the communication tree is constructed. The edge, which initiates the message sequences having sequence number 0, is made as root node of the communication tree. Next the edge having sequence number 1 becomes the child node of node 0. And if after that there are two edges numbered as 1.1 and 1.2 then the node 1 will have two children node one will be 1.1 and the other will be 1.2. After constructing the communication tree their next step is now to traverse the communication tree to select the conditional predicate. This technique traverses the Communication tree in post ordered manner. While traversing the tree in post order for selecting the condition predicate the predicate which are in the leaf node will be selected first. After selecting the conditional predicate, functional minimization technique is applied. If the predicate is of the from (E1 op E2), where op is $\leq$, $<$, $\geq$, $>$ then F (predicate function) = (E1-E2) or (E2-E1) depending upon which is positive. This process is done to test the boundary testing. The final step is to generate the Test Data. While generating the test data one set of data should be generated so that the predicate function becomes true and another set of test data should be generated for which the predicate function becomes false. In this step while generating the test data the advantage of post order traversal process comes. Since this technique first takes the leaf node conditional predicate so while generating test data for a predicate function if any pre condition path is there then it should satisfy that resulting less number of test data.

Sarma et al. [2] proposed an approach of generating the test sequence from UML Sequence Diagram. The method consists of two steps:

a) First the Sequence Diagram (SD) is converted into an intermediate format called Sequence Diagram Graph (SDG).
b) Secondly the SDG is traversed to generate the test cases.

First the SD is converted into SDG. The SD is not enough to decide the different component for test case generation. So, every node in the SDG contains the necessary informations for test case generation. These informations are obtained from

OCL (Object Constraint Language). For constructing the SDG from the SD first the set of operation scenarios are derived from the SD then depending upon the operation scenarios the SDG is constructed. An operation scenario represents a set of messages passed between the objects during each operation. Simply an operation scenario can be defined as a quadruple, aOpnScn: <ScnId; StartState; MessageSet; NextState>. A ScnID is a unique number which identifies each operation scenario. StartState is a starting point of the ScnId, that is, where a scenario starts. The set of all events that occur in an operation scenario is denoted MessageSet. The state that a system enters after the completion of a scenario is represented by NextState this is the end state of an activity or a use case. It may be noted that an SDG has a single start state and one or more end state depending on different operation scenarios. An event in a *MessageSet* is denoted by a tuple, *aEvent*: <*messageName*; *fromObject*; *toObject [/guard]*> where, *messageName* is the name of the message with its signature, *fromObject* is the sender of the message and *toObject* is the receiver of the message and the optional part */guard* is the guard condition subject to which the *aEvent* will take place. An *aEvent* with * indicates it is an iterative event. Depending upon the operation scenario the SDG is constructed. When ever a message is passed from one object to another object the state is changed from S1 to S2 and from S2 to S3. So while constructing the SDG the states are taken as node in the SDG and an edge is assigned between two nodes. Then the SDG is traversed using a traversal algorithm to generate the test cases. Since the Sequence Diagram represents the various interactions possible among the objects. So the test set derived for the SD should be able to detect whether the right sequences of messages are followed or not (Scenario fault) and the fault occur when an object invoke the method of another object (Interaction fault).

Sarma et al. [3] presents a method for generating test cases from UML Sequence Diagram and Usecase Diagram. The approach consists of following steps:

a) First the Usecase Diagram (UCD) is converted into a graph called Usecase Diagram Graph (UDG),
b) Next the Sequence Diagram is converted into a graph called Sequence Diagram Graph (SDG).
c) Finally the UDG and the SDG is integrated to generate a graph called System Testing Graph (STG). The STG is then traversed to generate the test cases.

While constructing a UDG from UCD all the actor of the UCD is replaced by a node in the UDG. A directed edge is connected between each pairs of nodes to represents the dependency between the nodes. The SD is converted into SDG using the same technique used by M. Sharma et al [2]. The SDG and UDG are combined to form STG. The STG is being traversed to generate the test cases.

Swain et al. [4] proposed a method to generate the test cases from UML Sequence Diagram and Usecase Diagram. Generating test sequences from Usecase Diagram (UD) consists of following three steps:

a) First the UD is converted into an Activity Diagram (AD) to specify the sequential constraint (sequential constraint is the order in which activity are to be maintained).
b) Then the Activity Diagram is being converted into Usecase Diagram Graph (UDG).
c) Test sequences are generated from the UDG.

While constructing the AD from the UD the nodes of the AD represent the use cases, and the edge represents the sequential dependency between the use cases. There may be use cases which are not sequentially related. They are represented by fork and join bar in the AD. In the next step while constructing UDG from the AD the nodes of the UDG represent the action node of AD, and the edge represents the dependency edge of AD. The fork and join bar of the AD are removed in the UDG. For generating use cases sequence the AD is traversed in Depth-First-Search (DFS) to find out the possible paths from the AD. After finding the test sequences the each node of the AD is mapped with the UDG to find out the corresponding use case test sequences.

Deriving the test sequences from the Sequence Diagram (SD) consist of three steps. First the SD is converted into an Activity Graph (AG) using a mapping rule. Then a Concurrent Control Flow Graph (CCFG) is being constructed from the AG. In CCFG a node represents the Activity node and an edge represents the control transfer among the nodes. Next all Concurrent Control Flow Paths (CCFP) are identified from the CCFG by finding out all the possible paths from the start node to end node of the CCFG. These paths are the Sequence Diagram test sequences. Finally the test cases are obtained by combining the test sequences obtained from the Usecase diagram and the test sequences obtained from the Sequence Diagram.

Nayak et al. [5] proposed a method for generating test cases from UML Sequence Diagram (SD). This approach consists of enriching the SD with attribute and constraint informations derived from OCL (Object Constraint Language). The SD is being map into a Structured Composite Graph called SCG. The test specifications are generated from SCG. This technique uses the UML 2.0 SD. The SD available in UML 2.0 uses an important feature called *Combined Fragment (CF).*A CF combines multiple operation scenarios. A CF may also contain another CF within it. This mechanism enables complex scenarios to be specified in a single sequence diagram. There are 13 different types of CF available in UML 2.0 such as ALT (choice of activities), PAR (group of activity to be executed parallel) etc. To show the control flow unambiguously the SD is converted into an intermediate format called SCG. There are two types of node in the SCG one is Block Node and another is the Control node. A block node in SCG is a node corresponding to a set of messages from the Sequence Diagram. Since a fragment is expected to alter the flow of control, a control node is used to mark the entering and leaving of a fragment. There four types of control nodes are there. A decision node is used for displaying the selection behaviour. A merge node is used for displaying exit from the selection behaviour. A set of fork and join node is used as an entry and exit from a par fragment. In SCG the Block nodes which represents the sequence of messages within one fragment of the SD is represented by oval shaped node and only the node-id is mentioned for each nodes. The guard associated to a fragment is shown as an edge descriptor. For representing the decision and merge node of the SD diamond shaped are used and for representing the fork and join of the SD thick line segments are used in the SCG. After constructing the SCG, the SCG is being traversed in Depth First Search manner for generating test scenarios.

Samuel et al. [6] proposed a method to generate test cases from UML 2.0 Sequence Diagram. This approach uses many of the novel features like alt, lop, opt etc of UML 2.0 Sequence Diagram. This technique consists of two main basic steps:

a) First the Sequence Diagram is converted into an intermediate format called Sequence Dependency Graph (SDG).
b) The SDG is being traversed to generate the test cases

This technique first defines the type of relationship that exists between the messages, to construct the SDG. There are four types of relationship that exist between two messages such as indirect message dependency, direct message dependency, simple indirect message dependency, simple direct message dependency. Based on the relationship the message sequences are generated. Message sequence is sequence of messages that are guaranteed to execute together. After finding out the type of relationship that exists between the messages the SDG is constructed. Each node in the SDG represents either a message or sequence of message. And edge is assigned between every pairs of nodes in the SDG. While constructing the nodes of SDG this technique uses the message number associated with each message instead of using the message name of the Sequence Diagram. The SDG is being traversed to find out the entire possible path from the start node to end node of the SDG in order to generate the test cases.

Lin et al. [7] proposed a method for generating test cases from UML Sequence Diagram (SD) and Object Constraint Language (OCL). This technique first constructs a Scenario Tree (ST) from the SD. While creating the ST from the SD the messages are represented by nodes and the sequence in which the messages are exchanged between the objects represents the edge. Then the ST is traversed using all message paths criterion (In case of all message paths criterion every message of the SD should be considered at least once) to iteratively select the attributes. The OCL is then used to store the pre and post conditions of each node. The attribute selection and the attribute transformation (representing pre and post conditions using OCL) are carried out till all the attribute are considered for test case generation. Using the pre and post conditions the test cases for that particular attribute is generated and then the next attribute is selected.

Santiago et al. [8] proposed a method for generating test case from Statechart and Finite State Machine. This technique is used to generate test cases for Implementation Under Test (IUT) projects and presents an environment called GTCS which enables the test sequence to obtain from both Statechart and FSM (Finite State Machine). GTCS stand for Geracao Automatica de Casos de Taste Baseada em Statecharts means Automated Test Case Generation based on Statechart. The test case is generated from the Statechart by following three basic steps:

a) First the Statechart is converted into a FSM.
b) Secondly a reachability tree is constructed from the FSM.
c) Finally a set of test case is generated from the FSM.

This technique first transforms the Statechart into FSM. Then the reachability tree is constructed from the FSM. A reachability tree shows the possible configurations and paths (sequence of configurations) that the system can reach. After obtaining the reachability tree the tree is traversed using all transition coverage criteria to generate the test cases. All transition coverage criterion state that the generate test case should encounter all the possible transition of the reachability tree.

2) For concurrent systems

Bader et al. [9] proposed a method for generating test cases for concurrent systems from UML Statechart Diagram. This technique takes Statechart Diagram as input and converts it into a tree. The tree is being traversed to generate the test case. In their approach they have taken an example of Telephone operator system which receives and forwards the calls. They have converted the Statechart of the telephone operator system into an event tree where each node of the event tree represents events of the Statechart. The tree (Event tree) is traversed in DFS (Depth-First-Search) to generate the test cases. Simply the test cases can be generated by parsing the branches of tree from the root node to each leaf node. So, the number of test cases for a Statechart is equal to the leaf node of the tree representing Statechart.

Various works [10,11,12,13] has been done for generating test cases for concurrent system using UML Activity Diagram. Kundu et al. [10] proposed a novel approach of generating test cases from UML Activity Diagram. The approach consists of following two steps:

a) First Activity Diagram (AD) is converted into Activity Graph (AG).

b) After that the AG is traversed to generate the test cases.

The AD is converted into an intermediate format called AG by using a mapping rule. While constructing the AG each activity of the AD is replaced by a node (one to one mapping) and an edge is assigned between two nodes of AG. In AG a node represents a state of doing something and an edge represents the flow between the activities. After constructing the AG the information about each node of AG is stored in a table called Node Description Table (NDT). The NDT maintain the information about each node of the AG, i.e. wether it is a fork node or a join node or a normal activity etc. After constructing the AG, the AG is traversed to generate the test cases. Every test case is generated using some coverage criteria and are aimed to detect certain faults. Test coverage criteria are a set of rules that guide to decide appropriate elements to be covered to make test case design adequate. In this approach Activity path coverage criterion is used. An activity path is a path in an AG which considers a loop at most two times and maintains precedence relationship between the activities. Each activity in AG is having at most one occurrences expect those activities which are in the loop, the activities in the loop are having at most two occurrences. Like every coverage criteria the activity path coverage criteria is aimed to detect three types of fault such as fault in decision, fault in loop, synchronization faults. Fault in a decision occur in the decision node of an activity diagram, for example in an activity diagram there is a decision node which decide the registration validity. Then there may situation where it may display the registration information of some registrant for some invalid registration id. Fault in loop occur in the entry or exit point of loop or increment, decrement operation. Suppose a loop is executed twice and at the end of iteration after giving try again = no, then instead of exiting from the loop the loop is executed for the third time. When some activity begins its execution before completion of execution of group of all preceding activities then synchronization faults occurs. Or simply synchronization faults occur when the concurrent preceding activities are not synchronized properly. The nonconcurrent activity path is used to find out the fault in the loop, and branch condition. And the concurrent activities are used to detect synchronized faults. Nonconcurrent activity path consist of set of sequential activities, concurrent activity path on the other hand consist of set of parallel activities. For generating test cases from the AG an algorithm is used called GenerateActivityPaths. The algorithm is a combination of DFS (Depth-First-Search), BFS (Breadth-First-Search). BFS is used to traverse the concurrent activities where as the rest activity are traversed using DFS. After applying the algorithm on the AG a set of Activity path are obtained. Then a rule is applied on the generated activity paths according to which each Activity path is decomposed into sequence of sub paths to obtain derived activity paths. Let AP_i is an activity path then decompose it into $AP_i = P_1\ P_i\ P_m\ P_i\ \ P_n$ if possible and then a derived path is obtain from AP_i which is $P_{derived}$(from AP_i) = $P_1\ P_i\ P_n$. The rule is applied on each activity path to replace decision/loop/fork-join blocks. To generate the test cases, after obtaining the activity paths and the derived activity paths now refer to NDT table to find out the information associated with each node. Constituent part of test cases are filled up by processing the paths and only taking into account when the node/s is/are object created, object state changed, sequence of branch condition, or activity sequences.

Sun [11] proposed a transformation-based approach for generating scenario oriented test cases for testing concurrent application by UML Activity Diagram. The approach consists of three basic steps:

a) First the UML Activity Diagram is transformed into an intermediate representation via a set of transformation rule.
b) Secondly from the intermediate format a set of test scenario is constructed.
c) Finally from the test scenario a set of test cases are derived.

This technique proposes a transformation rule to transform the Activity Diagram into an intermediate representation called Binary Extended AND_OR Tree (BET). The transformation rule is applied on fork node, join node, branch node and join node. For transforming the fork node with multiple out transition t_1, t_2, t_3 ……. ϵ T create a new node “n” in the ET, add a logic node FAND (ForkAND) in the BET. Connect the node “n” and FAND by an edge levelled with null. The out going transition of FAND are set as FAND – $e_1 \rightarrow n_1$,……. FAND – $e_m \rightarrow n_m$. The transformation of branch activity is similar to the fork activity however the FAND node is replaced by BOR (BranchOR). For transforming the join node with multiple in transition t_1, t_2, t_3 ……. ϵ T create a new node “n” in the BET and add logic node JAND (JoinAND) in ET. Connect the JAND and the “n” by an edge labeled with null. The in coming transition of JAND are set as $n_1 - e_1 \rightarrow$ JAND,……. $n_m - e_m \rightarrow$ JAND. The transformation of merge activity is similar to join activity but here the JAND is replaced by MOR (MergeOR). Where $n_1,\dots n_m$ are the mapped node activities of the AD and the $e_1 \dots e_m$ are mapped edged transition. This technique uses two separate tables to maintain the information about activity and the transition in the activity which are used in the further process of test case generation. Table 1 maintain the information associated with each activity of the Activity Diagram such as the activity number, activity types, details attribute of the activity, incoming transition and the out going transition of the activity.

Table 2 maintains the details information of each node and edge of the intermediate format, as well as the relation ship of each node in the intermediate format with the original activity. An algorithm is applied on the intermediate format to generate the Extended AND_OR Tree. The algorithm looks for Start/MOR/JAND nodes of the intermediate format and uses it as root node of ET. After that it took one by one node from the intermediate format and replaces it by corresponding nodes of BET. The traversal process is continued in a DFS (Depth-First-Search) manner until the leaf node of the tree (BET) under construction is a MOR/JAND/END. By applying the algorithm a set of trees are generated. After constructing the BET the BET is now traversed using a traversal algorithm, which traverse the BET according to weak concurrency coverage criteria to generate the test sequence from the BET for the original Activity Diagram. According to weak concurrency coverage criteria test scenario are derived to cover only one feasible sequence of parallel processes between a pair of fork and join activity, without considering the interleaving of activities between parallel processes. After obtaining the test scenario the set of test cases are obtained by referring to the tables to obtain the activity associated with each node of the intermediate format.

Kim et al. [12] proposed a method for generating test cases from UML Activity Diagram. This technique consists of following steps:

a) First built an I/0 explicit Activity Diagram from an ordinary UML Activity Diagram.
b) Then the I/O explicit Activity Diagram is converted into a directed graph.
c) From the directed graph the test cases for the original Activity Diagram is derived.

The I/O explicit Activity Diagram is an abstract representation of original Activity Diagram which is constructed by suppressing the non-external input and output and only showing the external input and output. Since the internal activities are having less importance than the input and output activities they are suppressed in IOAD to avoid the test case explosion. Then a directed graph is constructed from the I/O explicit Activity Diagram. The directed graph is traversed using DFS (Depth-First-Search) to generate a set of Basic paths, where each activity is considered once at a time since the technique uses basic paths (basic path coverage criterion considers each activity only once) coverage criterion to generate set of basic paths. In the next step a representative subset of path are selected from the set of basic path by removing the redundant edge and redundant nodes. The test cases are obtained from the representative set of paths.

Fan et al. [13] proposed a method for generating test cases from UML Subactivity and Activity Diagram. This technique consists of following steps:

a) Divides the Activity Diagram into Subactivity Diagram.
b) Then generate test cases for Activity Diagram from the Subactivity Diagram hierarchically.

The Subactivity Diagrams divides the Activity Diagram into Compound Activity Diagram (CAD) and Atomic Activity Diagram (AAD) which construct the hierarchy of whole Activity Diagram. In this technique first an Activity Diagram Composition Tree (ADCT) is constructed by taking the idea of functional decomposition to represent the hierarchy of Sub activity diagram and compound activity diagram. After obtaining the ADCT the test cases for the CAD is obtained by taking the idea of bottom-up testing strategy. According to bottom-up testing strategy first the leaf nodes of the ADCT are traversed, because each leaf nodes represents an atomic activity. Then the process is focused on Activity Diagram at the higher levels. This process is continued till the root node is encountered. For the activity diagram at the same level the generation order will be from left-right. After each activity diagram form different level in ADCT has its own test case set a round-robin method is used to generate the full test cases. The round-robin technique starts by picking the root node of activity diagram. Then retrieve every set of test cases of the root node, after that integrate it with the root node to generate the full test cases.

B) From Combination of UML Diagrams

Various works has been done to generate test cases from combinational UML models. Swain et al. [14] proposed a method for generating test cases from combination of UML Sequence Diagram and Activity Diagram. The technique consists of following steps:

1) First MFG is generated from Activity Diagram and Sequence Diagram.
2) In the second phase test sequences from MFG corresponding to sequence and activity diagrams is generated
3) In third phase the MFG of the Sequence Diagram and the MFG of the Activity Diagram is traversed to generate the test cases.

In the first phase the UML model are transformed into Message Flow Graph (MFG). The MFG can be represented as a quadruple (V, E, S, T) where each node $v \in V$ represents either a message or control predicate and an edge $e \in E$ represents a transition between the corresponding nodes. An edge (m, n) E indicates the possible flow of control from the node m to the node n. Nodes S and T are unique nodes representing entry and exit of the diagram D. For obtaining the MFG a table called Object Method Association Table (OMAT) table is created for the Sequence Diagram which maintains information about state change of an object when a message is passed between two objects. Another table is maintained for the Activity Diagram called Method Activity

Table (MAT) which maintains the activities associated with the Activity Diagram. By referring to the tables the MFG for the Activity and Sequence Diagram are created by taking each node and assigning an edge between them. The MFGs are next being traversed individually to generate the test cases.

Sokenou [15] proposed a method for generating test cases from combination of UML Sequence and State Diagram. In this technique the main information is obtained from Sequence Diagram. The Statechart diagram is attached in ordered to obtain the state information of each participating objects. Each Sequence Diagram specifies one test case or set of test cases. In this technique to obtain the required information of each node for test case generation the state diagrams are used. Each sequence diagram is considered as a set of test cases. An attached state diagram for each participating object defines its states. The sequence diagram is traversed and the information about each node is extracted out from the State Diagram to find out the test sequences.

III. DISCUSSION

The advantages of the technique proposed by Samuel et al. [1] are that it is helpful in detecting faults in cluster level testings as well as faults in boundary testing. In case of cluster level testing the interaction between the objects is tested. However the disadvantages of this technique are i) this technique orders every message using some sequence number, but if we will be using sequence diagram there is no need of doing such things ii) the technique generate the test data first for the leaf nodes by considering the pre path conditions of the leaf node. So, when ever the system is a large and complex system where large numbers of intermediate and leaf nodes are there to generate the test data for the conditional predicate of each leaf node we have to consider all pre paths conditions which is becomes time consuming and labour intensive. The advantage of the techniques proposed by Sharma et al. [2, 3] is that it is useful in detecting scenario as well as interaction fault. However the disadvantage of these techniques is that the techniques are not help full in detecting the decision faults. The advantage of the technique proposed by Swain et al. [4] is that it uses the important features of UML 2.0 Sequence Diagram such as interaction operand and constraints and combined fragment. This technique exercises object interactions in the context of use case dependencies to fulfil the requirements of the user. The advantage of the technique proposed by Nayak et al. [5] is that an effective set of test scenario are generated for a Sequence Diagram using this technique. However the disadvantage of this technique is determining of infeasible paths is a challenging task of this technique. Infeasible paths are those paths where there is no input data for them to be executed.This paths need to be detected and removed in order to generate optimized test scenario. The advantage of the technique proposed by Samuel et al [6] is that it uses the important features of UML 2.0 Sequence Diagram. However the disadvantage is that for constructing the SDG it finds out the relationship exist between every pair of messages. This is a time complexity task when the system is it self a complex system and large number of messages are communicated between the objects. For example while constructing the SDG from the SD if the message is a reply message then it is not considered in construction process of SDG. And if there exist “is a part of” relation ship between the messages then they are grouped up to single node in SDG. The advantage of the technique proposed by Lin et al. [7] is that this technique generates effective test cases to achieve each message on link coverage as well as scenario faults. This technique also achieves pre and post conditions path coverage. However the disadvantage is that the generated test sequence will be large in numbers since we are using all message paths.

The advantage of the technique proposed by Santiago et al. [8] is that it provides a methodology for generating test cases for softwares that is Implementation Under Test (IUT). The advantage of the technique proposed by Bader et al. [9] is that the technique considers communications as well as concurrency issues while generating test cases for a concurrent system. However the disadvantage of [8, 9] is that test case explosion problem is the main issue of a Statechart due to consideration of each and every state that an object undergoes during its operations. The advantages of the technique used by Kundu et al. [10] is that it is capable of detecting the fault associated with the truth value of a do-while loop condition since this technique is using activity path coverage criterion where the loop is consider at most two. But this is not possible for the basic path coverage criterion because it consider the loop to be executed at most once. Secondly this approach is capable of detecting more faults like fault in loop, synchronization fault, as well as fault in decision than the previously existing approaches. Finally it is possible to find out the location of faults. However the disadvantage of this technique is that while generating the activity paths from the AD the paths where the loop is executed one or two times are taken, however there may be situation where the loop is executed for zero times. Those activity paths where the loop will be executed for zero times are not taken into considerations. So this technique is incapable of detecting the faults of the activity path where the loop is encountered zero times. The advantage of the technique proposed by Sun [11] is that it considers the parallel activities, as well as conditional activity of system for test case generation. It transforms the fork node, join node as well as the branch and join node to address the issue associated with concurrent and conditional activities. However the disadvantage of this technique is that it does not consider the loop conditions where certain activities needed to be repeated until some condition is satisfied. The advantage of the technique proposed by Kim et al. [12] is that it controls the test case explosion by suppressing the non-external input and out-put events. However the disadvantage of this technique is that it generates the test cases by basic-path coverage criterion, where each activity is having exactly one occurrence. That means the activity in loop will be executed exactly once. So when ever there will be a do-while loop then the fault associated with the true part of that loop can not be detected as explained in [10]. The second disadvantage of this technique is that while generating set of representative path from the basic path it removes the redundant nodes and redundant edges, but what are the redundant nodes and edges has not explained here. The advantage of the technique Fan et al. [13] is that it uses the Round-robin-strategy to generate the test cases, which generate less number of test cases as compare to complete combination strategy. However here the disadvantages is that since here each activity is decomposed into a set of sub activities so, by applying this technique the number of generated test cases will be more as compare to the techniques that generate the test cases from normal activity diagram. The advantage of the technique proposed by Swain et al. [14] is that it uses the combination of UML Activity and Sequence Diagram. So, using this technique it is possible to detect faults associated with Activity Diagram as well as the faults associated with the Sequence Diagram. The advantages of the technique proposed by Sokenou [15] is that since it uses the combination of Sequence and Statechart diagram it is possible to obtain the set of message sequence along with the pre and post condition associated with each objects. However using the Statechart results in test case explosion as in the case of [8, 9].

IV. CONCLUSION AND FUTURE WORK

In this paper we have gone through a literature survey on generating test cases from UML models. Various works that has been done on generating test cases from single UML model or combinational models for concurrent as well as nonconcurrent system has been presented. Here we have discussed the techniques used in different article for generating test cases, their advantages, and disadvantages. This paper will help researcher to find out what work has been done in their interested field.

In future we are planning to generate test cases from combinational UML models. Where we will take different UML models, after converting the UML models into their intermediate formats we will combine the intermediate formats and will generate the test cases from the combinational intermediate format. We will also prove that our technique is capable of detecting more number of faults than compared to single UML models.

As explained in the Fig.1 we will first convert the System Models into intermediate formats called System Graph by using a mapping algorithm. The individual System Graphs will be combined using an integration algorithm which will integrate the individual graphs to form a Combinational System Graph. The Combinational System Graph will be traversed using a traversal technique to generate the test cases. Since the Combinational Graph will be formed by combination of more than one UML models so, it will have more coverage area and high fault detection capability than compared to the single UML models.

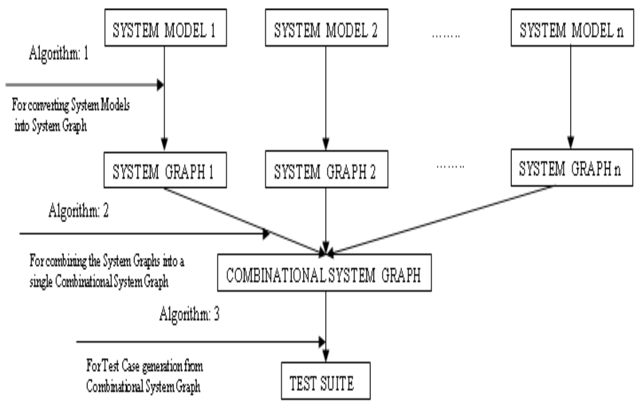
SYSTEM MODEL 1
SYSTEM MODEL 2
....
SYSTEM MODEL n
Algorithm: 1
For converting System Models into System Graph
SYSTEM GRAPH 1
SYSTEM GRAPH 2
....
SYSTEM GRAPH n
Algorithm: 2
For combining the System Graphs into a single Combinational System Graph
COMBINATIONAL SYSTEM GRAPH
Algorithm: 3
For Test Case generation from Combinational System Graph
TEST SUITE

FIG.1 Frame work for generating test cases from combinational UML Models

REFERENCES

[1] P.Samuel, R.Mall, P. Kanth, "Automatic test case generation from UML communication diagram", Information and Software Technology, Elsiver, 2007, pp. 158 – 171.

[2] M. Sharma, D. Kundu, R.Mall, "Automatic Test Case Generation from UML Sequence Diagrams", 15th International Conference on Advanced Computing and Communications, IEEE, 2007, pp. 60 – 65.

[3] M. Sharma, R.Mall, "Automatic Test Case Generation from UML Models", 10th International Conference on Information technology, IEEE, 2007, pp. 196 – 201.

[4] S. K. Swain, D. P. Mohapatra, R. Mall "Test Case Generation Based on Use case and Sequence Diagram", International Journal of Software Engineering, IJSE Vol.3 No.2 July 2010 pp. 21 – 52.

[5] A. Nayak, D. Samanta "Automatic Test Data Synthesis using UML Sequence Diagrams", in *Journal of Object Technology*, vol. 09, no. 2, March{April 2010, pp. 75 – 104.

[6] P. Samuel, A. T. Joseph, "Test Case Generation from UML Sequence Diagrams", Ninth ACIS International Conference on Software Engineering, Artificial intelligence, Networking, and parallel/distributed computing, IEEE, 2008 pp. 879 – 887.

[7] L. B. Lin, L. Z. Shu, L. Qing, C. Y. Hong "Test Case Automate Generation from UML Sequence diagram and OCL Expression" International Conference on Computational Intelligence and Security IEEE, 2007, pp. 1048 – 1052.

[8] V. Santiago, N. L. Vijaykumar, D. Guimaraes, A. S. Amaral, E. Ferreira "An Environment for Automated Test Case Generation from Statechart-based and Finite State Machine-based Behavioural Models ", IEEE 2008, pp. 1 – 10.

[9] A. Bader, S. M Sajeev, S. Ramakrishnan. "Testing Concurrency and Communication" in Distributed Objects.

[10] D. Kundu, D.Samanta, "A Novel Approach to Generate Test Cases from UML Activity Diagrams", Journal of Object Oriented Technology, vol. 8, no 3, May-June 2009 pp. 65 – 83.

[11] C. Sun, "A Transformation-based Approach to Generating Scenario-oriented Test Cases from UML Activity Diagram for Concurrent Applications", Annual IEEE International Computer Software and Applications Conference, IEEE, 2008, pp. 160 –167.

[12] H. Kim, S. Kang, J. Baik, I. Ko, "Test Cases Generation from UML Activity Diagram", Eight ACIS International Conference on Software Engineering, Artificial intelligence, Networking, and parallel/ distributed computing, IEEE, 2007 pp. 556 – 561.

[13] X. Fan, J. Shu, L. Liu, Q. Liang, "Test Case Generation from UML Subactivity and Activity Diagram", Second International Symposium on Electronic Commerce and Security, IEEE. 2009, pp. 244 – 248.

[14] S. K. Swain, D. P. Mohapatra, "Test Case Generation from Behavioral UML Models", International Journal of Computer Applications, Volume 6– No.8, September 2010, pp. 5 – 11.

[15] D. Sokenou, "Generating Test Sequences from UML Sequence Diagrams and State Diagrams" pp. 236 – 240.