Journal of Theoretical and Applied Information Technology
15th November 2021. Vol.99. No 21
© 2021 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

GENERATION OF COMBINATORIAL LOGIC ORIENTED TEST CASES FROM UML SEQUENCE DIAGRAM

SUBHASH TATALE[1]*, Dr. V. CHANDRA PRAKASH[2]

[1] Research Scholar, [2] Professor
Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, AP, India
*Corresponding author E-mail:subhashtatale@gmail.com

ABSTRACT

In the current practice, the test cases are generated from UML artefacts depending on the experience of testers in testing. Many researchers used different techniques to generate test cases from UML artefacts. There is a need to generate combinatorial logic-oriented test cases for those systems where combinatorial logic is necessary.
Combinatorial testing plays an essential role in generating a minimum number of the test cases to detect defects caused by interactions among system parameters. To generate combinatorial logic-oriented test cases, information about parameters, their values, and constraints is essential.
UML Sequence Diagram represents the dynamic behaviour of a software system. Extracting and identifying information about parameters, values and constraints from UML Sequence Diagram and detecting interactions among those extracted parameters is challenging task. The authors proposed multi-stage algorithm to extract and identify information about parameters, values and constraints from Sequence Diagram. The authors designed and developed a technique that automatically generates combinatorial logic-oriented test cases from UML Sequence Diagram.
In this paper, a case study of Concession Management SubSytsem of Indian Railways is presented. The authors generated automated test cases using the proposed Combinatorial Logic Oriented Test Case Generator for the case study and compared those test cases with manually generated test cases. It is found that generated automated and manual test cases are matching the same with each other.

Keywords: *Covering Array, Combinatorial Test Case Generation, Behavioral UML Diagrams, Sequence Diagram, Railway Reservation System, Concession Management System*

1. INTRODUCTION

Generating test cases at design phase has several advantages over coding phase of Software Development Life Cycle (SDLC). The test cases generated in the design phase remain valid even when we do a little bit change in the code [1]. The design models which are designed in design phase can be used as an input for generating the test cases. This will help to identify problems early in the stages of SDLC, which significantly reduces the time and cost of testing. However, generating test cases from Unified Modelling Language (UML) model is difficult task. UML has now become the de facto standard for object-oriented modelling and design. UML models are an essential source of information for generating test cases [2]. An automatic generation of test cases from UML model is a practically essential and receiving more awareness from researchers. Many researchers have presented various methodologies and techniques to generate test cases automatically using UML Sequence

Diagrams [3]. There are many systems like Concession Management SubSystem (CMSS) of Indian Railways, College admission system, Tuition fee concession subsystem, etc., in which combinatorial logic is extensively used. Combinatorial Testing (CT) is gaining high importance to test such type of systems.

The Combinatorial Test Design Model (CTDM) is popularly used to generate combinatorial test cases automatically [4]. There is a need to generate combinatorial test cases for those systems where combinatorial logic is essential. Different UML diagrams are used to model these systems. The same UML diagrams are used to generate combinatorial test cases.

The primary challenge of generating combinatorial logic-oriented test cases from the UML Sequence Diagram is to identify the input parameters, associated values, and constraints.

Many times application fails because of an interaction among the values of the different parameters of that application. It is an error-prone and challenging task to identify the input parameters, values and constraints manually. Hence, there is a need to model these parameters so that it reduces error and increase quality of the software system. A multi-stage algorithm is proposed to extract parameters, values and constraints from the sequence diagrams.

In this paper, the authors presented a case study of CMSS of Indian Railways to generate combinatorial logic-oriented test cases. The requirement specifications of CMSS are firstly modelled in the UML Sequence Diagram, and then all parameters, their values and constraints are extracted from this diagram using proposed multi-stage algorithm. After extraction of this information, combinatorial logic rules are applied and combinatorial test cases are generated. In the next section, concepts related to Sequence Diagram and Combinatorial Test Case Generation is discussed.

1.1 Combinatorial Test Case Generation

Grindal et al. [5] and Nie et al. [6] did a detailed survey on CT. The authors covered all aspects of CT including test modelling and CT applications. If the number of parameters and the values of these parameters are large, then it is practically impossible to generate all the combinatorial test cases. Covering Array (CA) which is a mathematical approach is used to reduce the test cases is [7] [8]. CA has four parameters, namely the number of input parameters (p), number of values (v), interaction strength (t) and number of test cases (N) [9].

For example, input has three parameters (namely X, Y, Z) having two values to two parameters each and three values to one parameter. Total 3*2*2 = 12 test cases are required to test combinations of all these parameters and values.

Table 1: System inputs having parameters and values

	Parameters		
	X	**Y**	**Z**
Values	X1	Y1	Z1
	X2	Y2	Z2
	X3	-	-

The optimal test suite has only six tests by using pairwise testing as shown in Table 2.

Combinatorial testing does not cover all parameter combinations, but it does show significant results in terms of detecting maximal defects in a small test suite. Table 2 shows a reduction in the size of the test suite from 12 to 6. Although it may not be exciting, we may test the effectiveness by using a more complex test input. Consider a test input with 20 parameters and ten values per parameter. 10^{20} test cases were developed by exhaustive testing. We may minimise the test suite size to 213 test cases by using combinatorial testing.

Table 2: An optimal test cases using pairwise testing strategy

Test case number	**X**	**Y**	**Z**
1	X1	Y1	Z1
2	X1	Y2	Z2
3	X2	Y1	Z1
4	X2	Y2	Z2
5	X3	Y1	Z1
6	X3	Y2	Z2

Generating combinatorial logic-oriented test cases is one of the test case generation techniques that focus on covering combinations of parameters, their values and constraints. The parameters are categorized into input/output parameters, constraints, and an infeasible combination between parameters and values. Deriving the CTDM is a necessary and critical step in the process of creating combinatorial test cases. CTDM consists of the elements like parameters, values of those parameters and constraints among the parameters and their values. Extracting these parameters and values is an innovative process that cannot be completely automatic.

1.2 UML Sequence Diagram

UML Sequence Diagrams are used to capture dynamic behaviour of a system from a different perspective. Sequence Diagram visualizes time-dependent interaction among the objects. It depicts the message sequence, as well as their names, responses, and probable counter arguments. In a Sequence Diagram, the vertical line shows time, whereas the horizontal line shows interaction among different objects [10].

In UML 2.0, different interaction fragments are used to describe many traces compactly and concisely. A fragment is an interaction operator which is used to showcase a conditional flow in the Sequence Diagram. It operates on a group of operands. Each operand represents a sequence of messages that occur under a guard condition. Some of the interaction operators are explained in Table 3.

Table 3: Interaction operators with its purpose

S. No.	Operator Name	Purpose
1	alt (alternatives)	This operator is used using multiple operands to capture alternative flows
2	opt (optional)	This operator has only one operand that is interpreted optionally
3	Break	This operator is used to capture an exit pathway of the systems
4	Loop	Thisoperator is used to model the repetitive interactions in a diagram.
5	neg (negative)	Thisoperator describes a combined fragment of traces that are defined to be negative (invalid).

The rest of the paper is structured as follows: In Section 2, the related work is briefly discussed. Section 3 explains the proposed work. Section 4 describes the results ad findings of the proposed work, while Section 5 contains concluding remarks.

2. RELATED WORK

The related work of test case generation and combinatorial test design model from the Sequence Diagram is discussed in this section. Also, a case study of Concession Management SubSystem of Indian Railways is discussed.

2.1 Test Case Generation from Sequence Diagram

Many researchers published research articles on the test case generation from Sequence Diagram using different approaches. Subhash Tatale et al. [13] published a survey paper on Test Case Generation using UML Diagrams and Feasibility Study to Generate Combinatorial Logic Oriented Test Case. The authors covered various test case generation techniques from Sequence Diagram.

Using the Formal specification approach, Panthi Vikas et al. [14], Zhang Chen et al. [15], RhmannWasiur et al. [16], and Nour El Houda Dehimi et al. [17] developed test cases. For this, the authors used Model checking, Formal specification, Object Constraint Language (OCL), and an Agent-based approach. Message and Path coverage criteria are met using these methods and approaches.

Using the Graphical representation approach, Samuel Philip et al. [18], Swain et al. [19] [20], and Dhineshkumar, M et al. [21] developed test cases. The methods of dynamic slicing and iterative deepening Depth First Search are employed. This technique meets the path and full predicate coverage criteria.

Jena Ajay Kumar et al. [22] employed a heuristic approach to generate test cases from a Sequence Diagram automatically. For this, a genetic algorithm is applied, and it meets the message coverage criteria.

For producing test cases, Beyer et al. [23] and Costa Leandro et al. [24] employed the Direct UML specification processing approach. The Markov Chain Usage Model and Parsing method are used to achieve a message coverage condition.

The Concurrent model approach was used by Khandai Monalisha et al. [25] [26] [27] and Mani P. et al. [28] to build test cases. Message, Sequence, and Path coverage criteria are met using Depth First Search, Breadth First Search, and Stack array approaches.

2.2 Combinatorial Test Design Model from Sequence Diagram

The related work on the combinatorial test design model is discussed in this section. Sasi Bhanu et al. [29] [30], Mudarakola et al. [31] and M. Laxmi Prasad et al. [32] [33] published research articles for testing distributed embedded systems using combinatorial testing methods. V. Chandra Prakash et al. [34] [35] performed a review on automated generation of combinatorial test cases using particle swarm optimization and generated test cases for pairwise + testing. For safety-critical embedded systems, Vudatha et al. [36] [37] [38] used a genetic algorithm to derive combinatorial test cases from the output domain. This method ensures that the largest numbers of output combinations are thoroughly generated and tested. In a constraints management context, Ramgouda Patil et al. [39] [40] [41] proposed a Neural Network strategy to improve combinatorial coverage, as well as multi-objective crow search and fruit-fly optimization strategies to optimize combinatorial test cases.

There is a need to generate combinatorial logic-oriented test cases from UML artefacts. Satish Preeti et al. [42] presented a rule-based semi-automated approach for obtaining the information of combinatorial test design model. Sajad Esfandyari et al. [43] employed model checking approaches to extract parameters and values from state space. Subhash Tatale et al. [44] proposed an approach of enhancement in acceptance test-driven development using combinatorial logic.

2.3 Concession Management SubSystem – A Case Study

In this section, the authors presented a case study of CMSS of Indian Railways. Indian Railways offers concessions on ticket fares to different categories of concessions like Disabled Passenger, Patient, Senior Citizen, Child, War Widow, etc. These concessions are offered based on various types of journey classes like Sleeper Class, First Class, etc. There are several types of concessions in each concession category.

Subhash Tatale et al. [44] published a research article on applying Combinatorial Logic to improve the Acceptance Test Driven Development Model. The authors presented Software Requirement Specification (SRS) of CMSS of Indian Railways in the view of combinatorial logic. While generating combinatorial test cases for the mentioned journey classes (7 journey classes), concession categories (11 concession categories) and their types (174 concession types) in the research article [44], the size of test cases may be enormous because of too many combinations of parameters and values in the input. If we apply All Combinations testing technique to those concession categories and types, then the total number of generated test cases will be 2x3x15x127x5x5x6x3x3x3x2= 92583000. It is challenging and unrealistic to generate and to test such a large number of test cases. It is called a Combinatorial Explosion problem of test cases.

Therefore, the authors of this paper considered limited journey classes and concession categories to avoid the combinatorial explosion problem of test cases. The authors condensed some of the concession categories and types in the revised SRS of the CMSS. Only limited Journey class, concession categories and types are considered in the revised SRS of the CMSS. The list of different concession categories, types as per revised SRS of CMSS is shown in Table 4. These requirements are considered to generate combinatorial logic oriented test cases from the Sequence Diagram.

Table 4: List of different categories of concessions along with % of concession

Category of Passenger	**Reservation Class**	
	Sleeper	**First**
	Percentage of Concession	
Disabled Passenger		
Handicapped	75	75
Mentally retarded	75	75
Patient		
Cancer	100	75
Heart	75	75
Passenger Type		
Senior Citizen (>= 60 years)	50	50
Child (<=12 years)	50	50
Widow		
War Widow	75	50

Figure 1 depicts Sequence Diagram of CMSS considering revised SRS as per Table 4. Figure 1 is shown at the end of the paper.

3. THE PROPOSED COMBINATORIAL LOGIC-ORIENTED TEST CASE GENERATOR (CLOTCG)

It is a very tedious and challenging task to identify manually the exact number of parameters, their values, and constraints from UML diagrams. It is very difficult to extract the information in manual way when the values of the parameters are dynamic. In this section, the authors presented a technique that helps to extract the required preliminary information automatically from the UML Sequence Diagram in the form of parameters, values and constraints. The combinatorial logic is applied to that extracted information to generate combinatorial logic-oriented test cases.

The authors show how to generate combinatorial logic-oriented test cases using the suggested Combinatorial Logic-Oriented Test Case Generator (CLOTCG) technique from UML Sequence Diagram. A model is designed based on Software Under Test (SUT) and then the elements of CTDM are generated. The multi-stage algorithm is used to extract the information from those CTDM. Combinatorial logic is used to construct the combinatorial logic-oriented test cases.

Figure 2 depicts the Data Flow Diagram of the proposed CLOTCG. The Sequence Diagram is drawn using the StarUML tool as per the requirement specifications. A test manager will give Sequence Diagram as an input to the proposed system. The information like parameters, values and constraints are extracted from Sequence Diagram using multi-stage algorithm. Sequence diagram is converted into XML Metadata Interchange (XMI). The XMI document is extracted using JavaScript Object

Notation (JSON) object. JavaScript is used to parse JSON object for validation of the extracted information of the Sequence Diagram. Once it is validated, then the information required for the CLOTCG is extracted. The combinations of parameters and values are generated from extracted parameters and values. The constraints are processed on a combinatorial generated list. After that, combinatorial logic-oriented rules are applied. Finally, combinatorial logic-oriented test cases are generated. The proposed technique for extracting useful information from a Sequence Diagram is explained in detail in the following section.

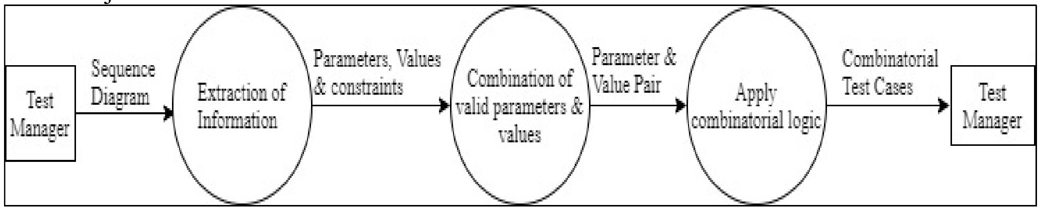
Test Manager
Sequence Diagram
Extraction of Information
Parameters, Values & constraints
Combination of valid parameters & values
Parameter & Value Pair
Apply combinatorial logic
Combinatorial Test Cases
Test Manager

Figure 2. Data Flow Diagram of the proposed CLOTCG

3.1 Extraction of information from Sequence Diagram

The information like parameters and values is essential to generate combinatorial logic-oriented test cases. In this section, the authors presented a technique that extracts the information which is essential to generate combinatorial logic-oriented test cases. The UML sequence diagram is drawn using the StarUML tool.

3.1.1 Generate XML Metadata Interchange (XMI) from UML Sequence Diagram

XMI is a perceptive way of converting UML models into XML documents. It is used to exchange metadata information using Extensible Mark-up Language (XML). The main rationale for extracting XMI is to allow for a smooth interchange of metadata in distributed heterogeneous contexts between modelling tools and metadata repositories. In XMI format, the message, start and end of a fragment are first found out. The XMI Code is shown in Figure 3.

```
<uml:Model xmi:id="AAAAAAF5johXshGwltM=" xmi:type="uml:Model" name="RootModel">
<packagedElement xmi:id="AAAAAAFF+qBWK6M3Z8Y=" name="Model" visibility="public" xmi:type="uml:Model">
<packagedElement xmi:id="AAAAAAFzTuURyI4VdAI=" name="Collaboration1" visibility="public" isAbstract="false"
isFinalSpecialization="false" isLeaf="false" xmi:type="uml:Collaboration">
<ownedMember xmi:id="AAAAAAFzTuURyI4WSOI=" name="Interaction1" visibility="public" isReentrant="true"
xmi:type="uml:Interaction">
<lifeline xmi:id="AAAAAAFzTuXvBY4lDzQ=" name="Passenger" visibility="public"
xmi:type="uml:Lifeline"represents="AAAAAAFzTuXvBY4k9h4="/>
<lifeline xmi:id="AAAAAAFzTuYYgo5F4Bo=" name="Railway%20Authority" visibility="public" xmi:type="uml:Lifeline"
represents="AAAAAAFzTuYYgY5EVsQ="/>
<message xmi:id="AAAAAAF4WWSr/e8zOl4=" name="Select%20Journey%20Class" visibility="public" xmi:type="uml:Message"
messageSort="synchCall" messageKind="complete" receiveEvent="AAAAAAF5johXtRGy+t8="
sendEvent="AAAAAAF5johXtRGxCgO="/>
```

Figure 3. Code snippet of XMI

3.1.2 Generating JSON code from XMI

JSON is a lightweight data-transfer format for sending data between client and server that is simple to understand and generate. JSON, like XML, is a text-based format that is simple for humans and machines to write and interpret.

A JavaScript library called xml2json converts the given XML code to JSON code to generate a mapping of parameters and values.

This library acts as an XML to JSON

used to parse JSON object. This method is primarily used to return a JavaScript object from the string that is parsed. The string that will parsed with JSON.parse() is in a JSON format. The mapping of the parameter and values are stored in .csv file that is extracted through the JSON code. Figure 4 depicts the JSON code snippet.

```
{"uml:Model":{"$":{"xmi:id":"AAAAAAF5johXshGwltM=","xmi:type":"uml:Model","name":"RootModel"},
"packagedElement":{"$":{"xmi:id":"AAAAAAFF+qBWK6M3Z8Y=","name":"Model","visibility":"public","xmi:type":"uml:Model"},
"packagedElement":{"$":{"xmi:id":"AAAAAAFzTuURyI4VdAI=","name":"Collaboration1","visibility":"public","isAbstract":"false","isFina
lSpecialization":"false","isLeaf":"false","xmi:type":"uml:Collaboration"},
"ownedMember":{"$":{"xmi:id":"AAAAAAFzTuURyI4WSOI=","name":"Interaction1","visibility":"public","isReentrant":"true","xmi:type":
"uml:Interaction"},
"lifeline":[{"$":{"xmi:id":"AAAAAAFzTuXvBY4lDzQ=","name":"Passenger","visibility":"public","xmi:type":"uml:Lifeline","represents":"
AAAAAAFzTuXvBY4k9h4="}},
{"$":{"xmi:id":"AAAAAAFzTuYYgo5F4Bo=","name":"Railway%20Authority","visibility":"public","xmi:type":"uml:Lifeline","represents":
"AAAAAAFzTuYYgY5EVsQ="}}],
"message":[{"$":{"xmi:id":"AAAAAAF4WWSr/e8zOl4=","name":"Select%20Journey%20Class","visibility":"public","xmi:type":"uml:Mess
age","messageSort":"synchCall","messageKind":"complete","receiveEvent":"AAAAAAF5johXtRGy+t8=","sendEvent":"AAAAAAF5johXt
RGxCgQ="}},
{"$":{"xmi:id":"AAAAAAF4WWVFeO9KLkM=","name":"First%20or%20Sleeper","visibility":"public","xmi:type":"uml:Message","messag
eSort":"reply", "messageKind":"complete","receiveEvent":"AAAAAAF5johXthG0Ajk=","sendEvent":"AAAAAAF5johXthGz9WQ="}}
```

Figure 4. Code snippet of JSON

3.1.3 Extraction of parameters, values and constraints

1. Sequence Diagram with synchronous messages

The SUT input variables are considered as test input parameters. There are two object lifelines, namely Passenger called Object 1 and Railway Authority called Object 2. These two objects communicate using synchronous messaging. A solid arrowhead pointing from left to right denotes synchronous messaging. Every message that is sent receives a response message. This reply message is shown by a dotted arrowhead pointing from right to left. Object 1 will not be able to continue processing until it receives a response. As a result, this classification aids in the immediate identification of parameters and their values. The parameters and values shown in Figure 5 (a) are the primary information. This information is treated as an input of the system.

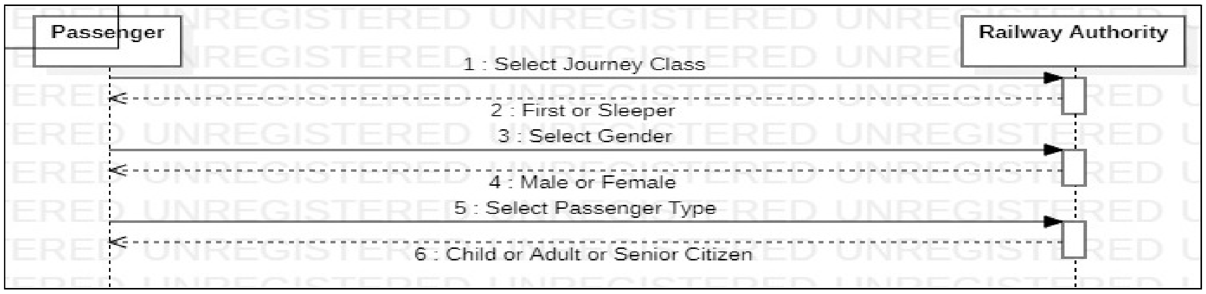

Figure 5. (a) Sequence Diagram with Synchronous Messages

The loop fragment is used to consider combinations of values for those parameters. The information shown in Figure 5 (b) and (c) are optional information. NS (Not Selected) indicates that the passenger has not selected any concession. 'and' keyword indicates that the passenger is considering multiple combinations of parameter values.

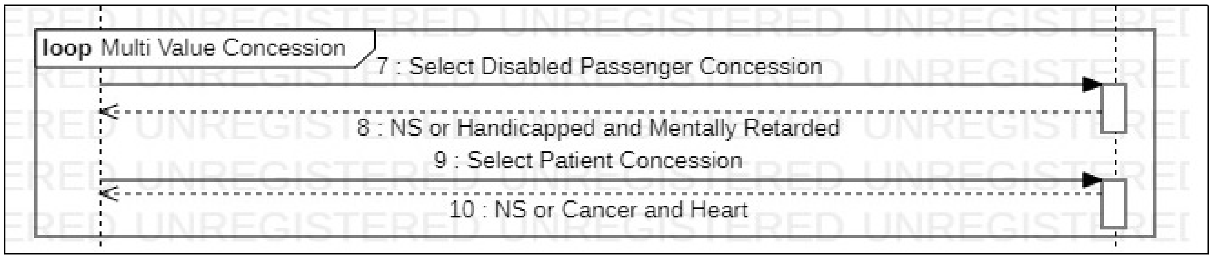
loop Multi Value Concession
7 : Select Disabled Passenger Concession
8 : NS or Handicapped and Mentally Retarded
9 : Select Patient Concession
10 : NS or Cancer and Heart

Figure 5. (b) Sequence Diagram with loop operator

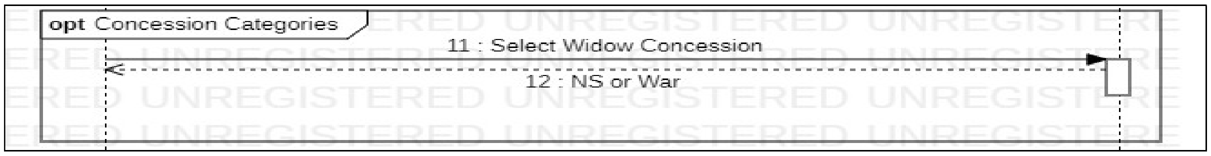

Figure 5. (c) Sequence Diagram with opt operator

a) Steps for parameter extraction

The outgoing messages from the lifelines of Object 1 to the Object 2 are called as parameters. The below steps are used to extract parameters from Sequence Diagram.

1: Read the JSON code

2: Get the object name of the Object 1 from the code

3: Find a message passed from the lifeline of Object 1 to the lifeline of Object 2.

4: Display the name of the message as a parameter.

b) Steps for value extraction

The incoming messages towards the lifelines of Object 1from the Object 2 is values. The below steps are used to extract values from Sequence Diagram.

1: Read the JSON code

2: Get the object name of the Object 1 from the code.

3: Find a message passed to the lifeline of Object 1 from the lifeline of object 2.

4: Display the name of the message as a value.

c) Steps for identifying constraints

The neg operator describes a combined fragment of traces that are defined to be negative or invalid.

The outgoing messages from the Object 1 to the Object 2 are combinations of parameters and values, and its reply message from the Object 2 back to the Object 1 is the infeasible combinations.

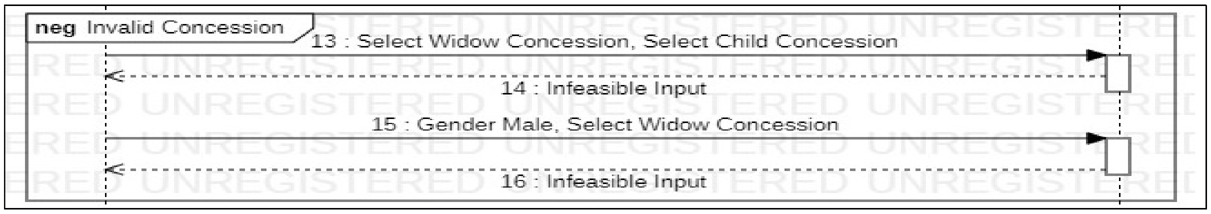
neg Invalid Concession
13 : Select Widow Concession, Select Child Concession
14 : Infeasible Input
15 : Gender Male, Select Widow Concession
16 : Infeasible Input

Figure 6. Sequence Diagram with neg operator

The below algorithm is used to identify the constraints from Sequence Diagram.

```
open the JSON code in read mode
if (combined fragment= neg)
{
    get the object name of the Object 1 from the code
    find a message passed from the Object 1 lifeline to the Object 2 lifeline
    if message is found
    {
        search next sibling message node of this message which is passed back to the Object 1 lifeline
        if message is Infeasible Input
        {
            display the message name
            map the infeasible combinations of parameters and their values respectively
        }
    }
```

Table 5 shows the extraction of constraints or invalid combinations of parameters and values after applying above mentioned steps on components of Sequence Diagram shown in figure 6.

Table 5:- Extracted constraints from Figure 6

Sr. No.	Concession Categories	Concession types	Infeasible concession categories
1	Gender	Male	Widow
2	Passenger type	Child	Widow

This information indicates that Male and Child passenger cannot be Widow and cannot avail Widow Concessions.

2. Identifying combinatorial logic oriented rules

In a sequence diagram, an alt operator is used to express a "if-then" condition. The outgoing messages from Object 1 to Object 2 are the combinatorial logic oriented concession rules, and the reply message from Object 2 to Object 1 is the percentage of concession offered to various parameter combinations.

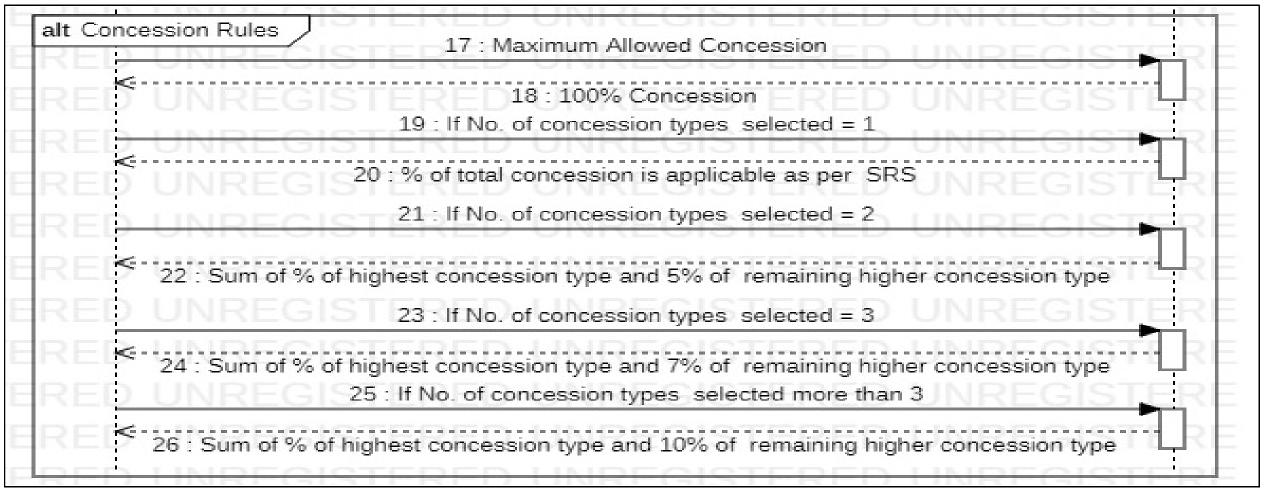
alt Concession Rules
17 : Maximum Allowed Concession
18 : 100% Concession
19 : If No. of concession types selected = 1
20 : % of total concession is applicable as per SRS
21 : If No. of concession types selected = 2
22 : Sum of % of highest concession type and 5% of remaining higher concession type
23 : If No. of concession types selected = 3
24 : Sum of % of highest concession type and 7% of remaining higher concession type
25 : If No. of concession types selected more than 3
26 : Sum of % of highest concession type and 10% of remaining higher concession type
UNREGISTERED

Figure 7. Sequence Diagram with alt operator

The below algorithm is used to identify the combinatorial logic rules from the JSON code.

```
1.  open the JSON code in read mode
2.  if (combined fragment= alt)
3.  {
4.      find a message passed from the Object 1 lifeline to the Object 2 lifeline
5.      if message is found
6.      {
7.          search next sibling message node of this message which is passed back to the Object 1 lifeline
8.          if message is found
9.          {
10.             display the value of the attributes of the messages
11.             return percentage of concession
12.         }
13.     }
```

Table 6 shows the combinatorial logic oriented concession rules from Figure 7.

Table 6:- Combinatorial logic oriented concession rules from Figure 7

Sr. No.	Concession rules as parameter	Total concession (in %) as value
1	If no. of selected concession types = 1	% of total concession is applicable as per Table 2.
2	If no. of selected concession types = 2	% of highest concession type + 5% of remaining concession type as per Table 2.
3	If no. of selected concession types = 3	% of highest concession type + 7% of remaining higher concession type as per Table 2.
4	If no. of selected concession types > 3	% of highest concession type + 10% of highest of the remaining concession type as per Table 2.
5	Maximum allowed concession	100%

3.2 Combination of parameters and values

In the previous section, the parameters and values are extracted from the Sequence Diagram using various steps. Combinations of parameters and values are necessary to generate combinatorial logic oriented test cases. In this section, the algorithm for

combination of extracted parameters and values is discussed. For the combination, mapping of appropriate parameters and values are necessary. The below algorithm is used for making combination of extracted parameters and values.

Algorithm:

```
read JSON code
for all parameter name from JSON code
{
    match message name with  parameter
    check sendEvent of that message
    match combined fragment with xmi:id
    check coverage of that fragment
    if coverage is found then
    {
        select xmi:id of the respective coverage
        match this xmi:id to the message of receiveEvent
        extract the value of corresponding parameter
    }
}
```

Table 7 shows the extracted parameters and associated values after applying above mentioned steps on components of Sequence Diagram shown in figure 5 (a) (b) (c).

Table 7:- Extracted parameters and values from Figure 5

Parameters	Journey class	Gender	Passenger type	Disabled Passenger	Patient	Widow
Values	1. Sleeper 2. First	1. Male 2.Female	1.Child 2.Adult 3.Senior Citizen	1. NS 2.Mentally retarded 3.Handicapped	1.NS 2. Cancer 3.Heart	1. NS 2. War

3.3 Apply combinatorial logic

The parameters, their associated values, constraints (if any) and combinatorial logic-oriented rules play an important role for generation of combinatorial logic-oriented test cases. We have covered the multi-stage algorithm techniques to extract this information in the previous section. The All Combinations (AC) testing technique generates every possible combination of parameters and values.

In the CMSS, we extracted parameters and values. We extracted six parameters like Journey Class, Gender, Passenger type, Disabled passenger, Patient, Widow and its associated values as per Table 4.

As per the Sequence Diagram shown in Figure 5 (b), the Disabled passenger and Patient category have multiple selection options. The multiple selection options are shown using AND conditional operator in figure 5 (b). It indicates that for Disabled Passenger and Patient category, multiple value combinations are selected.

Therefore, All Combinations test suite satisfies N-wise coverage.

All combination values= ∑ vi value combinations, where N is the coverage number (1-way, 2-way,...N-way) and vi is the number of values of parameter .

Hence, we can calculate all combination values for Disabled passenger and Patient parameter as follows:

All combination value (Disabled Passenger) = {Handicapped, Mentally retarded, Handicapped and Mentally retarded}

All Combinations value (Patient) = {Cancer, Heart, Cancer and Heart}

Therefore, values of Disables Passenger and Patient concession category are increased due to All Combinations testing techniques. All the remaining values of concession categories viz. Journey class, Gender, Passenger type and Widow are mutually exclusive. Table 8 shows number of values for each parameter after applying All Combinations testing technique.

Table 8:- Number of values for parameters using All Combinations testing technique

Parameters	Journey class	Gender	Passenger type	Disabled Passenger	Patient	Widow
No. of Values	2	2	3	4	4	2

Hence, total number of test case combinations are generated = 2x2x3x4x4x2=384 ---------(1)

There are some Infeasible Combinations are derived as shown in Table 5.

1. Male – Widow infeasible combinations
2. Child- Widow infeasible combinations

These combinations are considered as Not Applicable (NA). The NA value is assigned to respective parameter because of infeasible combinations between parameters and values. For such kind of test cases, NA and NS value conflicts to each other. These test cases are considered as duplicate test cases. The test cases containing NS value are removed as per below calculations.

1. Male – Widow infeasible combinations

Total number of duplicate test combinations because of Male- Widow infeasible combinations

= 2 (Journey Class) x 1(Gender) x 2 (Passenger type except Child) x 4 (Disabled passenger) x 4 (Patient) x 1(Widow) = 64 --------- (2)

2. Child- Widow infeasible combinations

Total number of duplicate test combinations because of Child- Widow infeasible combinations

= 2 (Journey Class) x 2 (Gender) x 1 (Passenger type - Child) x 4 (Disabled passenger) x 4 (Patient) x 1(Widow) = 64 ------------------(3)

Total number of duplicate test combinations generated because of Infeasible Combinations

= 64+64 (From Eq.2 and 3) = 128 -----------(4)

Total number of distinct test cases generated = 384-128 (from Eq. 1 and 4) = 256

Hence, total 256 test cases are generated from Sequence Diagram using CLOTCG technique automatically.

The percentage of concession is applied based on combinatorial logic oriented rules mentioned in Table 4 and 7. The test cases generated using the proposed CLOTCG is shown in Table 9.

Table 9:- Test cases generated using CLOTCG

TC No.	**Journey class**	**Gender**	**Passenger type**	**Disabled passenger**	**Patient**	**Widow**	**Expected concession (%)**
1	First	Male	Child	-	-	NA	50
2	First	Male	Child	-	Cancer	NA	77.5
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
256	Sleeper	Female	Senior Citizen	Handicapped and Mentally retarded	Cancer and Heart	War	100

These test cases also cover constraints of parameter-value combinations as per Table 6. The 'NA' value in the test case indicates all infeasible options. All combinations testing technique is good for the smaller size of parameter and values. If the number of parameters and values is large, it is a very tedious task to generate and test those exhaustive test cases.

4. RESULTS AND DISCUSSIONS

The authors presented All Combinations testing technique to generate combinatorial logic-oriented test cases from UML Sequence Diagram. These test cases will aid in improving the testing efficacy. All Combinations testing technique gives a better result for UML Sequence Diagram. It is found that generated automated and manual test cases are matching the same with each other. The Input Size of Table 8 indicates the number of parameters and values extracted from the UML Sequence Diagram.

CA ($2^1 2^1 3^1 4^1 4^1 2^1$) input size indicates 6 concession categories (parameters) having some number of concession types (values). Out of 6 parameters, 3 parameters are having 2 values each, 1 parameter is having 3 values and 2 parameters are having 4 values each. CA ($2^2 3^2 4^2 15^1$) input size indicates 7 concession categories (parameters) having some number of concession types (values). Out of 7 parameters, 2 parameters are having 2 values each, 2 parameters are having 3 values, 2 parameters are having 4 values and 1 parameter is having 15 values.

The result of the proposed technique is shown in Table 10.

Table 10:- Results of proposed techniques

Sr. No.	Input Size	No. of Test cases generated manually	No. of Test cases generated by CLOTCG	Accuracy (%)
1	CA($2^1 2^1 3^1 4^1 4^1 2^1$)	256	256	100
2	CA($2^2 3^2 4^2 15^1$)	2208	2208	100

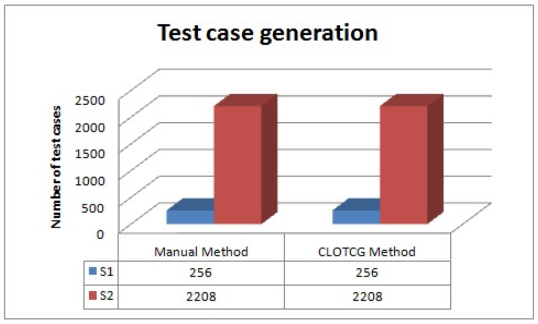

Figure 8. Result comparison of Manual method and proposed CLOTCG method

Figure 8 shows the result comparison of number of test cases generated by Manual method and proposed CLOTCG method. It shows that proposed method gives the 100% accuracy.

4.1 Findings

Generating a large number of test cases is a very laborious, time-consuming and costly task

by using manual method. In addition, during manual test case generation, some erroneous test cases may be generated. The test cases are currently designed by the test designers from UML artefacts based on their experience with the manual testing approach.

The authors of this paper found a research gap for generating combinatorial test cases automatically from UML artefacts. An automatic test case generation is essential to avoid erroneous test cases generated using the manual testing approaches. Moreover, automatic generation of test cases can reduce testing cost by eliminating costly manual test case generation. Thus, it reduces the time and cost of testing significantly.

The automatic test case generation from UML diagrams will help to identify problems in the early stages of SDLC. The test cases generated from the UML Sequence Diagram using the proposed CLOTCG are 100% accurate. The accuracy percentage shows a comparison between manually generated test cases and automated generated test cases using the CLOTCG. The authors claimed that the proposed testing technique gives reliability and efficiency ultimately.

4.2. Extension to Existing Techniques

Many researchers proposed different approaches like Formal specification-based approach, Graphical representation approach, Heuristic approach, Direct UML specification processing approach, Hybrid behaviour model approach and Concurrent model approach to generate test cases from UML Sequence Diagram. These approaches are used to generate the test cases for functional testing of the software systems.

There is a need to provide combinatorial logic-oriented test cases for the systems that use combinatorial logic, such as reservation systems, college entrance systems, concession management systems, and so on. The authors of this paper suggested a novel method for generating combinatorial logic-oriented test cases from UML Sequence Diagrams.

The proposed technique generates combinatorial test cases based on parameters, values, and constraints extracted from Sequence Diagram, whereas the existing techniques generate test cases based on paths, messages, sequences, etc.

4.3 Limitations of the Proposed Technique

UML represents software requirements specification in a graphical or diagrammatic way. Sometimes, it is challenging to represent all these requirements using UML diagrams. Some diagram shows behavioural, creational, structural views of the requirements. Any single diagram cannot capture all the requirements of software systems. Hence, it is a limitation of the generation of test cases from one of the UML diagrams. It is infeasible and challenging to generate combinatorial test cases for complex Sequence Diagrams.

Only Sequence Diagram cannot capture all the requirements of the software system. Therefore, some requirements might get missed out and subsequently, those test cases will not get generated. As a result, combinatorial test cases must be built from other UML diagrams such as Activity Diagram, Use Case Diagram, and State Machine Diagram in order to capture all requirements and generate test cases.

5. CONCLUSION

Using the manual testing technique for generating a large number of test cases is an extremely time-consuming, labor-intensive, and expensive task. Furthermore, certain incorrect test cases may be generated during manual test case generation. The test cases are currently being created by the test designers using UML artefacts and their manual testing experience. The authors found that there is need to generate combinatorial test cases automatically from UML artefacts to avoid erroneous test cases which are generated using the manual testing approaches.

In this paper, the authors proposed CLOTCG technique to generate combinatorial logic-oriented test cases from Sequence Diagram automatically. The authors presented a multi-stage algorithm to extract parameters, values and constraints from UML Sequence Diagram. The guard conditions of various combination fragments and the messages in synchronous message calls of the Sequence Diagram are used to identify these parameters, values, and constraints. The appropriate rules are applied for various semantic constructs, guard condition, synchronous message calls, and constraints of Sequence Diagram.

CMSS of Railway Reservation System of Indian Railways is presented as a case study to demonstrate the proposed technique. The authors generated automated test cases using the proposed Combinatorial Logic Oriented Test Case Generator for the case study and compared those test cases with manually generated test cases. It is found that generated automated and manual test cases are matching the same with each other.The authors claimed that the proposed testing technique gives reliability and efficiency completely.

In the future, combinatorial logic-oriented test cases can be generated to capture all the requirement specifications from other UML diagrams like Activity Diagram, State Chart Diagram, etc.

REFERENCES

[1] Hartmann, J., Vieira, M., Foster, H. and Ruder, A., 2005. A UML-based approach to system testing. Innovations in Systems and Software Engineering, 1(1), pp.12-24.

[2] Briand, L. and Labiche, Y., 2002. A UML-based approach to system testing. Software and Systems Modeling, 1(1), pp.10-42.

[3] Nebut, C., Fleurey, F., Le Traon, Y. and Jezequel, J.M., 2006. Automatic test generation: A use case driven

approach. IEEE Transactions on Software Engineering, 32(3), pp.140-155.

[4] Lott, C., Ashish Jain, and S. Dalal. "Modeling requirements for combinatorial software testing." In ACM SIGSOFT Software Engineering Notes, vol. 30, no. 4 (2005), pp. 1-7.

[5] Grindal, Mats, Jeff Offutt, and Sten F. Andler. "Combination testing strategies: a survey." Software Testing, Verification and Reliability 15, no. 3 (2005): 167-199.

[6] Nie, Changhai, and Hareton Leung. "A survey of combinatorial testing." ACM Computing Surveys (CSUR) 43, no. 2 (2011): 1-29.

[7] Kuhn, D. Richard, Renee Bryce, Feng Duan, Laleh Sh Ghandehari, Yu Lei, and Raghu N. Kacker. "Combinatorial testing: Theory and practice." In Advances in Computers, vol. 99, pp. 1-66. Elsevier, 2015.

[8] R. Kuhn, Yu Lei and Raghu Kacker, “Practical Combinatorial Testing: beyond Pair wise”, IEEE Computer Society - IT Professional, Vol. 10, No. 3 (2008).

[9] D. Richard Kuhn, Raghu N. Kacker and Yu Lei, “Practical combinatorial testing”, NIST Special Publication, (2010).

[10] Shirole, M. and Kumar, R., 2013. UML behavioral model based test case generation: a survey. ACM SIGSOFT Software Engineering Notes, 38(4), pp.1-13.

[11] F. Basanieri, A. Bertolino, and E. Marchetti, “The Cow Suite Approach to Planning and Deriving Test Suites in UML Projects," in Proceedings of the 5th International Conference on The Unified Modeling Language, UML '02, pp. 383-397, Springer-Verlag, 2002.

[12] S. Ali, L. Briand, H. Hemmati, and R. Panesar-Walawege, “A systematic review of the application and empirical investigation of search-based test case generation," IEEE Transactions on Software Engineering, vol. 36, no. 6, pp. 742 -762, 2010.

[13] Subhash Tatale, Dr. V. Chandraprakash. " A Survey on Test Case Generation using Sequence & Activity diagrams and a Feasibility Study to generate Combinatorial Logic Oriented Test Cases", International Journal of Next-Generation Computing - Special Issue, Vol. 12, No. 2, April 2021. pp.254-269

[14] Panthi Vikas and Durga Prasad Mohapatra. "Automatic test case generation using sequence diagram." In Proceedings of International Conference on Advances in Computing, pp. 277-284. Springer, New Delhi, 2013.

[15] Zhang Chen, Zhenhua Duan, Bin Yu, Cong Tian, and Ming Ding. "A test case generation approach based on sequence diagram and automata models." Chinese Journal of Electronics 25, no. 2 (2016): 234-240.

[16] Rhmann Wasiur, and Vipin Saxena. "Test Case Generation from UML Sequence Diagram for Aadhaar Card Number based ATM System." system 11, no. 4 (2016).

[17] Dehimi Nour El Houda, and Farid Mokhati. "A Novel Test Case Generation Approach based on AUML sequence diagram." In 2019 International Conference on Networking and Advanced Systems (ICNAS), pp. 1-4. IEEE, 2019.

[18] Samuel Philip and Rajib Mall. "A Novel Test Case Design Technique Using Dynamic Slicing of UML Sequence Diagrams." e-Informatica 2, no. 1 (2008): 71-92.

[19] Swain, Santosh Kumar, and Durga Prasad Mohapatra. "Test case generation from Behavioral UML Models." International Journal of computer applications 6, no. 8 (2010): 5-11.

[20] Swain Ranjita Kumari, Vikas Panthi, Prafulla Kumar Behera, and Durga Prasad Mohapatra. "Slicing-based test case generation using UML 2.0 sequence diagram." International Journal of Computational Intelligence Studies 2 3, no. 2-3 (2014): 221-250.

[21] Dhineshkumar, M. "An approach to generate test cases from sequence diagram." In 2014 International Conference on Intelligent Computing Applications, pp. 345-349. IEEE, 2014.

[22] Jena Ajay Kumar, Santosh Kumar Swain, and Durga Prasad Mohapatra. "Test case creation from UML sequence diagram: a soft computing approach." In Intelligent Computing, Communication and Devices, pp. 117-126. Springer, New Delhi, 2015.

[23] Beyer, Dulz and Fenhua Zhen, "Automated TTCN-3 test case generation by means of UML sequence diagrams and Markov chains," 2003 Test Symposium, Xi'an, China, 2003, pp. 102-105.

[24] Costa, L.T., Zorzo, A.F., Rodrigues, E.M., Silveira, M.B. and Oliveira, F.M., “Structural Test Case Generation Based on

System Models". In Proceedings of the 9th International Conference on Software Engineering Advances pp. 276-281, 2014.

[25] Khandai Monalisha, Arup Abhinna Acharya and Durga Prasad Mohapatra. "A novel approach of test case generation for concurrent systems using UML Sequence Diagram." In 2011 3rd International Conference on Electronics Computer Technology, vol. 1, pp. 157-161. IEEE, 2011.

[26] Monalisa Sarma Debasish Kundu Rajib Mall. "Automatic test case generation from UML sequence diagrams." In 15th International Conference on Advanced Computing and Communications, pp. 60-65. 2007.

[27] Sarma Monalisa, and Rajib Mall. "Automatic test case generation from UML models." In 10th International Conference on Information Technology (ICIT 2007), pp. 196-201. IEEE, 2007.

[28] Mani P., and M. Prasanna. "Test case generation for embedded system software using UML interaction diagram." Journal of Engineering Science and Technology 12, no. 4 (2017): 860-874.

[29] Dr.Sasi BhanuJ, Dr.Baswaraj D, Sunitha Devi Bigul, Dr. JKR Sastry, Generating Test cases for Testing Embedded Systems using Combinatorial Techniques and Neural Networks based Learning Model, International Journal of Emerging Trends in Engineering Research, Volume 7, No. 11 November 2019, pp 417-429.

[30] J. Sasi Bhanu, M. Lakshmi Prasad, Dr. JKR Sastry, A Combinatorial Particle Swarm Optimization (PSO) Technique for Testing an Embedded System, Jour of Adv Research in Dynamical & Control Systems, Vol. 10, 07-Special Issue, 2018, pp. (321-336).

[31] Mudarakola, Lakshmi Prasad, J. K. R. Sastry, and V. Chandra Prakash."Testing embedded systems using test cases generated through combinatorial techniques." International Journal of Engineering & Technology 7, no. 2.7 (2018): 146-158.

[32] M. Lakshmi Prasad, Dr. J Sasi Bhanu, , Dr. J. K. R. Sastry, Combinatorial Neural Network Based a Testing of an Embedded System, Jour of Adv Research in Dynamical & Control Systems, Vol. 10, 07-Special Issue, 2018.

[33] M. Lakshmi Prasad, A. Raja Sekhar Reddy, J.K.R. Sastry, GAPSO:Optimal Test Set Generator for Pairwise Testing, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8 Issue-6, August 2019.

[34] Dr.V.Chandra Prakash, Subhash Tatale, Vrushali Kondhalkar, Laxmi Bewoor. "A critical review on automated test case generation for conducting combinatorial testing using particle swarm optimization." International Journal of Engineering & Technology (UAE), Vol.7, No.3.8, (2018), pp. 22-28.

[35] V.Chandra Prakash and Kadiyala Priyanka, 2016. "Test Case Generation for Pairwise + Testing." Asian Journal of Information Technology. Vol.15 No.23 (2016), pp.4800-4805.

[36] Vudatha Chandra Prakash, Sastry K R Jammalamadaka, and Bala Krishna Kamesh Duvvuri. "Automated generation of Test cases for testing critical regions of embedded systems through Adjacent Pair-wise Testing." International Journal of Mathematics and Computational Methods in Science & Technology Vol.2, No.2, (2012), pp. 10-15.

[37] Vudatha, Chandra Prakash, Sateesh Nalliboena, Sastry Kr Jammalamadaka, Bala Krishna Kamesh Duvvuri, and L. S. S. Reddy. "Automated generation of test cases from output domain of an embedded system using Genetic algorithms." 3rd International In Electronics Computer Technology (ICECT), IEEE (2011), vol. 5, pp. 216-220.

[38] Vudatha, Chandra Prakash, Sateesh Nalliboena, Sastry KR Jammalamadaka, Bala Krishna Kamesh Duvvuri, and L. S. S. Reddy. "Automated generation of test cases from output domain and critical regions of embedded systems using genetic algorithms." 2nd National Conference on Emerging Trends and Applications in Computer Science, pp. 1-6. IEEE, 2011.

[39] Ramgouda Patil, V Chandra Prakash, "Neural Network Based Approach for Improving Combinatorial Coverage in Combinatorial Testing Approach", Journal of Theoretical and Applied Information Technology, Vol.96. No 20 (2018),pp. 6677-6687

[40] Gouda, Ram, and V. Chandraprakash. "Optimization Driven Constraints Handling in Combinatorial Interaction Testing." International Journal of Open Source Software and Processes (IJOSSP) 10, no. 3 (2019): 19-37.

[41] Ramgouda, P., and V. Chandraprakash. "Constraints handling in combinatorial interaction testing using multi-objective crow search and fruitfly optimization." Soft Computing 23, no. 8 (2019): 2713-2726.

[42] Satish Preeti, Arinjita Paul, and Krishnan Rangarajan. "Extracting the combinatorial test parameters and values from UML sequence diagrams." In 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops, pp. 88-97. IEEE, 2014.

[43] Esfandyari, Sajad, and Vahid Rafe. "Extracting Combinatorial Test parameters and their values using model checking and evolutionary algorithms." Applied Soft Computing 91 (2020): 106219.

[44] Subhash Tatale, Dr. V. Chandraprakash, "Enhancing Acceptance Test Driven Development Model with Combinatorial Logic", International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 11, No. 10, 2020. Pp.268-278

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

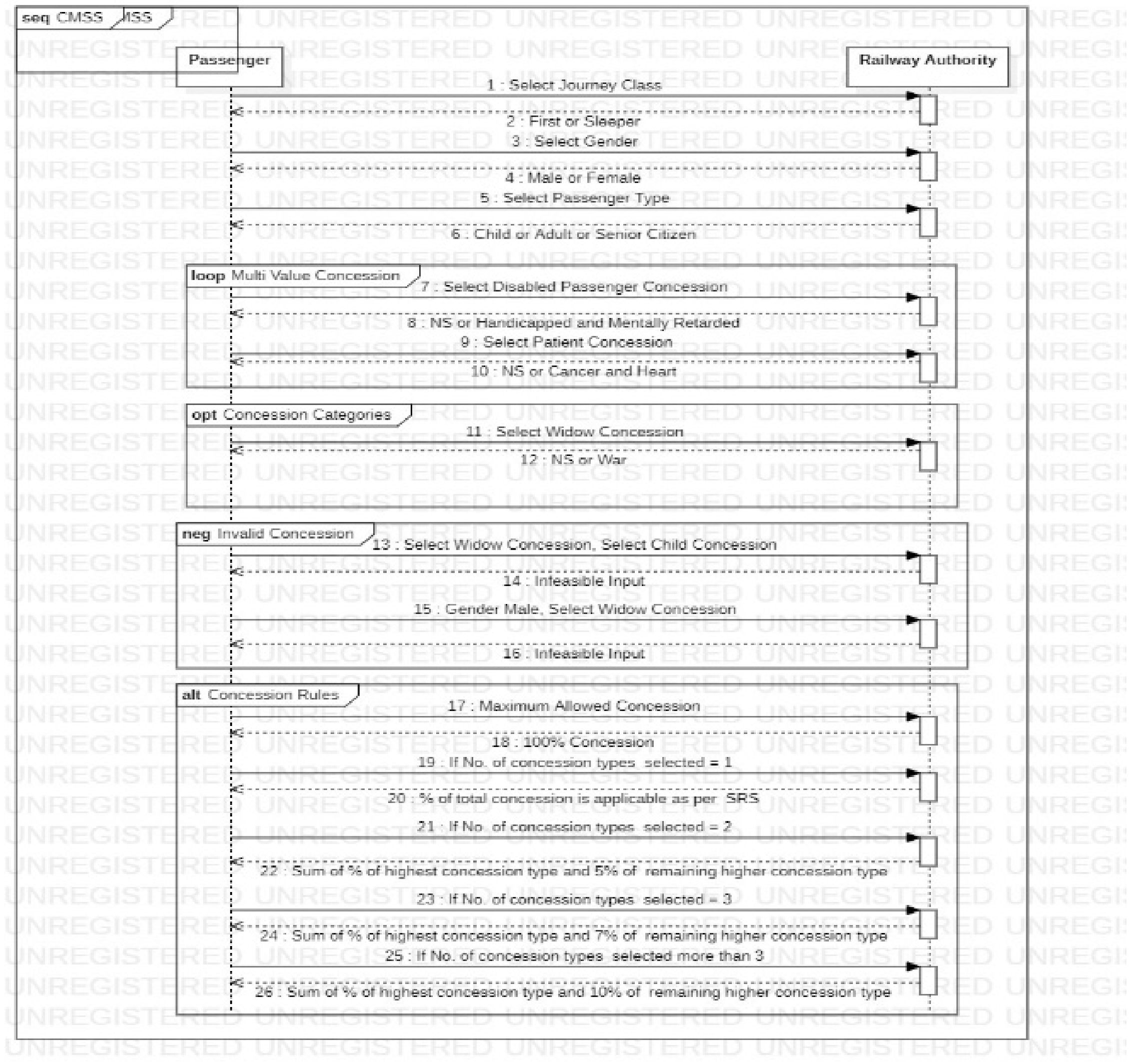
seq CMSS
Passenger
Railway Authority
1 : Select Journey Class
2 : First or Sleeper
3 : Select Gender
4 : Male or Female
5 : Select Passenger Type
6 : Child or Adult or Senior Citizen
loop Multi Value Concession
7 : Select Disabled Passenger Concession
8 : NS or Handicapped and Mentally Retarded
9 : Select Patient Concession
10 : NS or Cancer and Heart
opt Concession Categories
11 : Select Widow Concession
12 : NS or War
neg Invalid Concession
13 : Select Widow Concession, Select Child Concession
14 : Infeasible Input
15 : Gender Male, Select Widow Concession
16 : Infeasible Input
alt Concession Rules
17 : Maximum Allowed Concession
18 : 100% Concession
19 : If No. of concession types selected = 1
20 : % of total concession is applicable as per SRS
21 : If No. of concession types selected = 2
22 : Sum of % of highest concession type and 5% of remaining higher concession type
23 : If No. of concession types selected = 3
24 : Sum of % of highest concession type and 7% of remaining higher concession type
25 : If No. of concession types selected more than 3
26 : Sum of % of highest concession type and 10% of remaining higher concession type

Figure 1. Sequence Diagram of the revised CMSS